



Engineering Virtual Domain-Specific Service Platforms

Specific Targeted Research Project: FP7-ICT-2009-5 / 257483

Requirements Engineering Framework, Language and Tools for Service Platforms

Abstract

This document presents the methodologies, languages, and tools proposed in INDENICA for the holistic elicitation of requirements. The document is split into four parts: Part 1 is an introduction, Part 2 explains the Requirement Engineering process derived from Product Line Engineering and transferred to Platform as a Service, Part 3 consists of two INDENICA specific adaptations: IRENE, the goal-based solution for stating the actual requirements of the service platform and the User Centered Requirements Engineering Approach, Part 4 provides the tool support for both methods presented in part 3 (IRET and a prioritization tool).

Document ID:	INDENICA – D1.2.2
Deliverable Number:	D1.2.2
Work Package:	1
Type:	Deliverable
Dissemination Level:	Public
Status:	Final
Version:	1.0
Date:	2012-10-01
Contributing Partners:	SAP, SIE and PDM

Version History

0.1	12 September 2012	Initial version
0.2	20 September 2012	PDM contribution added
0.3	27 September 2012	SAP contribution added
04.	30 September 2012	Revised version
1.0	1 October 2012	Final version

Document Properties

The spell checking language for this document is set to UK English.

Table of Contents

1	Introduction	6
2	RE Process and Methods for Platforms as a Service	7
2.1	RE for Domain Engineering	8
2.1.1	Requirements Management.....	8
2.1.2	Requirements Development	9
2.2	RE for Application Engineering.....	21
2.2.1	Requirements Management.....	21
2.2.2	Requirements Development	22
2.3	Relation to Agile Methodologies	23
2.4	Application of process on virtual service platforms	27
2.5	ROI Calculation	28
3	INDENICA specific notations	35
3.1	IRENE.....	35
3.2	Views.....	35
3.3	Variability support	36
3.4	Model integration.....	37
3.4.1	Semantic similarity.....	38
3.4.2	Requirements matching.....	39
3.5	User Centered Requirements Engineering	40
3.5.1	Motivation: Summary of work presented in D1.2.1	40
3.5.2	Graphical display of Priorities.....	40
3.5.3	Analysis of Conflicting Sets of Prioritization.....	41
4	Tools.....	44
4.1	IRET	44
4.1.1	Improvements on Attributes, Events and Entities.....	45
4.1.2	Views	46
4.1.3	Merge	49
4.1.4	Other improvements.....	55
4.1.5	Integration with other tools	56
4.2	Prioritization Tool Support	59
4.2.1	Goal Extractor	59

4.2.2	Prioritization Analyzer	60
5	Conclusions	65
6	References.....	66
	Appendix	69

List of Acronyms

AE	Application Engineering
DE	Domain Engineering
EMF	Eclipse Modelling Framework
GMF	Graphical Modelling Framework
IRENE	INDENICA Requirements Elicitation mEthod
IRET	IREne Toolset
LTL	Linear Temporal Logic
PaaS	Platform as a Service
PLE	Product Line Engineering
RE	Requirements Engineering
PO	Product Owner
ROI	Return On Invest

1 Introduction

This document is the continuation of D1.2.1 and presents the work done over the last year on the methodologies, languages, and tools proposed in INDENICA for the holistic elicitation of requirements. The document discusses the RE process and methods for PaaS, the INDENICA specific methods like ROI calculation, IRENE, and UCRE (user-centric RE), and the tool support.

Besides the work already presented in the first deliverable, the document concentrates on the following aspects:

- A more comprehensive requirements elicitation process and methods for platforms as a service, with emphasis on both domain and application engineering;
- The introduction and analysis of agile methodologies and of User Story Mapping (USM) in particular;
- An improved and complete solution for ROI calculation, as means to estimate the costs associated with this kind of solutions and decide about the feasibility of the project;
- The new features introduced in IRENE to consider service platforms explicitly: views, view merging, variability modelling, semantic similarity, and matching;
- The methodology defined for integrating the different priorities set by the stakeholders involved in the elicitation process.
- The new version(s) of IRET, which is the IRENE supporting tool, to let the user try and exploit the new features of the language, and also to ameliorate some of the existing capabilities;
- The new tool developed to work on conflicting priorities and set them univocally.

The rest of the document is organized as follows. Section 2 presents the requirements elicitation process and methods for platforms as a service. Section 3 describes the novelties proposed by IRENE. Section 4 summarises the novelties in IRET and presents the new tool for working on conflicting priorities. Section 5 concludes the document.

2 RE Process and Methods for Platforms as a Service

As described in [SotA], the main characteristics for Product Line Engineering (PLE) consist in:

- The existence of two different development processes:
 - Domain Engineering (DE):
The process of software product line engineering, in which the commonality and the variability of the product line are defined and realized.
 - Application Engineering (AE):
The process of software product line engineering, in which the applications of the product line are built by reusing domain artefacts and exploiting the product line variability
- Variability as a core concept for PLE
- A platform for the product line
- A reference architecture

One main principle of PLE is building a platform. Figure 1 shows the position of Requirements Development within a complete PLE process. We take here the approach to identify methods of Requirements Engineering suitable for PLE and also for the development of a INDENICA virtual service platform.

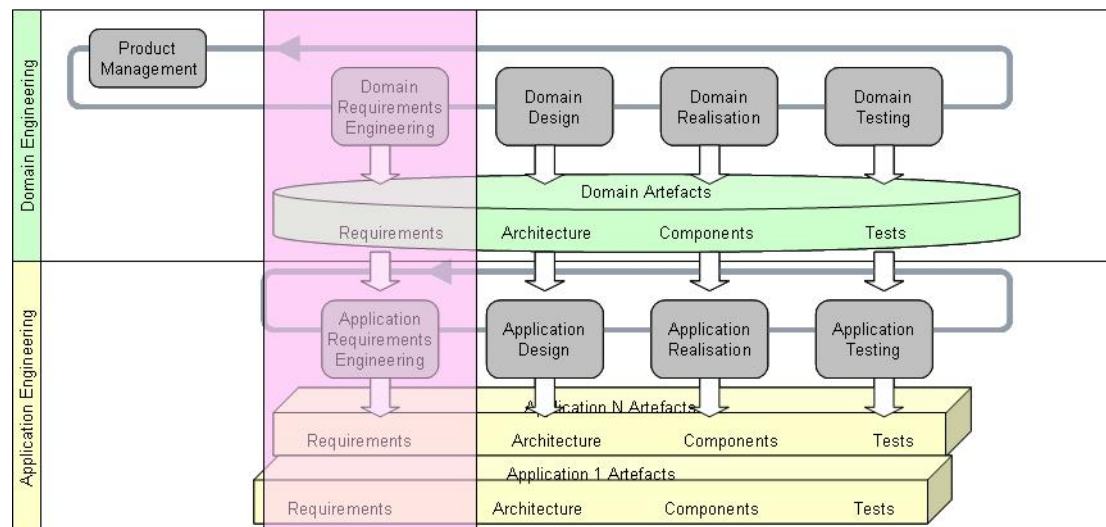


Figure 1: Requirements Development in Domain and Application Engineering.

For all following considerations regarding “Requirements Development in the context of PLE” we define the workflow in Figure 2 as basis for the process:

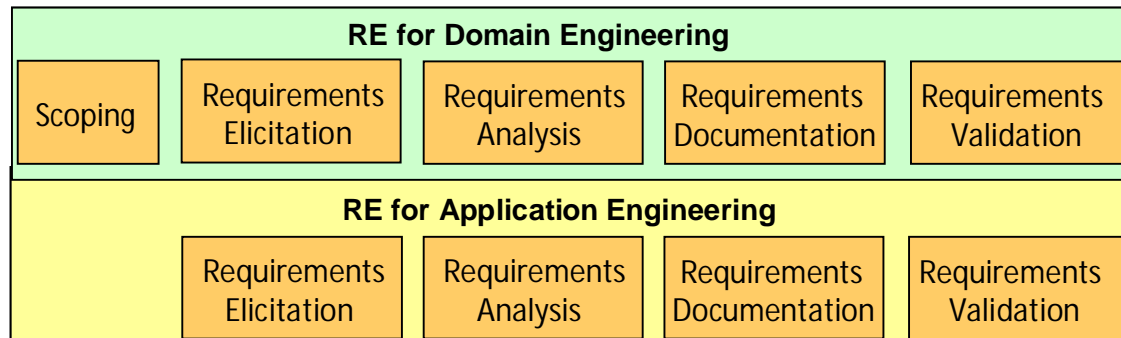


Figure 2: Requirements Development as process basis for PLE.

In the following sections the RE process for product lines is described in more detail, with challenges for RE arising from product line context.

2.1 RE for Domain Engineering

Domain Engineering is a specific discipline necessary when developing product lines. In the INDENICA context this discipline will be used to design the virtual service platform. The domain engineering process deals with the development of all parts of the product line that are common to all applications of the product line or a specific number of diverse applications of the product line. This has effects on different steps of the requirements engineering process.

2.1.1 Requirements Management

In general, the major task of Requirements Management is to manage all kinds of traceability (Figure 3):

- Tracing requirements back to their origin (1)
- Managing cross-references between requirements (2)
- Tracing requirements forward to their implementations (3)
- Managing requirements changes (4)

The main purpose of requirements management in the context of Domain Engineering is to ensure the consistence of requirements common to all related applications.

Thus traceability management is a crucial factor for product line engineering. Beside the tracing aspects relevant for product development, following challenges additionally have to be considered for tracing artefacts of product line engineering:

- Bi-directional tracing between platform and application artefacts (5)

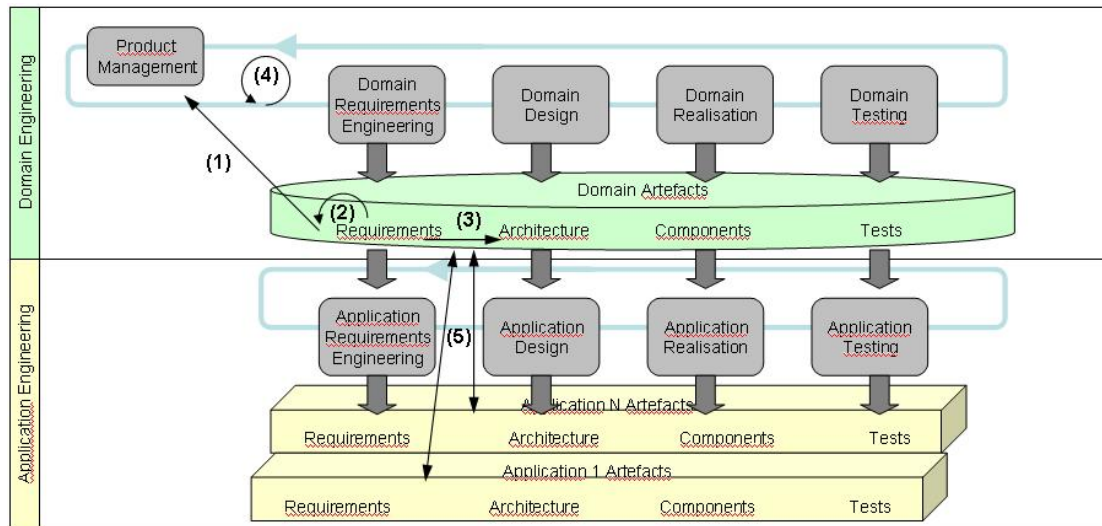


Figure 3: Platform Development Process Overview.

2.1.2 Requirements Development

The main purpose of requirements development in the context of Domain Engineering is to evolve the requirements common to all related applications or a well-defined subset of those applications and to document them in a structured manner.

All activities (already well-known for requirements development within product development) have to be fulfilled on a higher degree of complexity.

Additionally to those tasks (known for general requirements development), Domain Engineering requires an additional process step.

All these specific characteristics are described in the following chapter.

Scoping:

First of all it is crucial to define the scope for domain engineering (the platform system or core asset base and its boundaries).

Based on the results of the analysis for goals and strategies of the product portfolio (roadmap and strategy), the product line manager has to determine the scope for the platform.

Essential prerequisite for the scoping process are

- a commitment to business objectives
- a clear definition of the marketplace
- a product vision stretching out for the projected life of the product

These strategic decisions build the basis for the following scoping process.

Output of this process step is a “Product Line Strategy”, the commitment of aspects going into the product line, which is the base for the following Domain and Application Engineering Process.

Scoping is separated into three phases:

- Product Line Scoping
Identification and description of products within product line
- Domain Scoping
Identification of common and different areas
- Asset Scoping
Identification of core asset base

Methods:

Hereafter we will outline a range of suitable methods for scoping:

- ROI Calculation (see Section 2.5)

As „commitment to business objectives“ is an important aspect when introducing PaaS, it is crucial to estimate the “Return on Invest” (ROI) of such an approach.

The method described in chapter 2.5 uses a bunch of parameters:

- on the one hand, hard data like number of versions and applications
- on the other hand, estimated factors as cost for developing and using reuse assets with sophisticated formulas, which depend on the chosen environment (only SW development, SW+HW development, etc.).

A promising approach when taking into account all conceivable conditions constitutes a specific customization of the scoping process [John 2006].

The customization has to be applied according to different contexts:

- Operational Context
- Domain Characteristics
- Integratable Artefacts
- Enterprise Context
- Resources

This method specified in greater detail below bridges the gap between already existing scoping methods and the needs for ROI calculations oriented to these contexts.

- PuLSE-Eco (PuLSE process) see [PuLSe]

PuLSE-Eco is a method to define the scoping of a product line with emphasis on economical aspects.

This method separates the scoping process into three parts:

- Product Line Mapping

In the first part, the infrastructure for the following scoping activities will be set up.

This process step is primarily characterized by identification activities:

- all products relevant for the product line have to be identified and described,
- on the other hand, all technical domains and features have to be identified that could be serving as a basis for the products.

As input, following artefacts can be used: expert know-how, product plans, organizational structure and already existing systems.

The result of this process step consists of a high level description of the product line and their domains.

- Domain Scoping / Domain Potential Assessment

In the next step the identified domains have to undergo an evaluation of their domain potential.

The result of this process step consists of a range of promising candidates for common domains.

- Asset Scoping / Reuse Infrastructure Scoping

The last step of PuLSE-Eco covers a detailed analysis and identification of infrastructure assets meant for reuse resulting in a Product Map.

- Organization Domain Modelling (ODM)

ODM is another method for the scoping phase within domain engineering that has been developed by Mark Simos et al. in 1996 [STARS 1996].

It also consists of three major phases:

- Plan domain
- Model domain
- Engineer Asset Base

whereas the first two aspects belong to domain analysis and the last one ("engineer asset base") is already part of domain design and implementation. Hence, the two phases "Plan domain" and "Model domain" contribute especially to the scoping process, the first by defining and scoping the right domain, the second by describing and modelling the domain selected in the previous phase.

- Domain Scoping Framework

Another method for the scoping process is the Domain Scoping Framework developed in occasion of the workshop "Domain Analysis in the DoD" in 1995 sponsored by the Software Reuse Initiative (SRI).

Elicitation:

Process Steps:

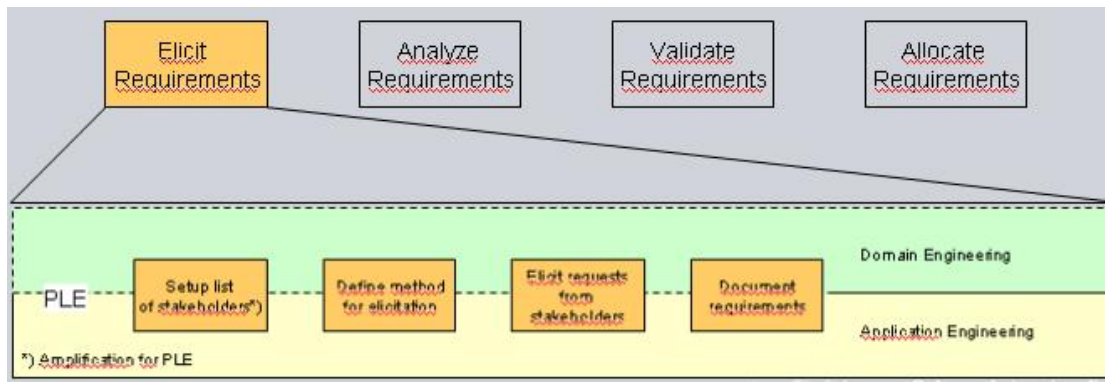


Figure 4: Requirements Elicitation Process.

The main challenges concerning requirements elicitation within domain engineering are [Pohl et al. 2005]:

- Identification of requirement sources for the whole product line
First challenge in eliciting requirements for the whole product line is to identify the range of requirements sources. A promising approach for identification is to consider all already existing applications and their requirements to the product line, as in most cases product lines are developed based on already existing applications.
As a product line may cover many diverse applications, this range of requirements sources possibly appears very heterogeneous and even contradictory. This leads to the next challenge:
- Identification of common requirements (commonality analysis)
The central benefit of PLE is the commonality of artefacts across diverse applications. For requirements engineering this implies the existence of common requirements for several applications.
 - A proven method for the evaluation of common requirements is the "application - requirements matrix". In this matrix it is listed which application is effected by which requirement. Requirements which appear for all applications may be identified as common requirements.
 - Another method is the priority-based analysis:
Different stakeholders are asked to prioritize a given set of requirements. A specific algorithm calculating the relevance of each requirement for the whole product line may give indication of the availability of common requirements.
- Identification of variable requirements (variability analysis)
The complementary analysis to commonality analysis is the variability analysis. It is performed using the same methods as commonality analysis:
 - application - requirements matrix
 - priority based analysis
Requirements which are assigned only to a few applications or are high-rated only by a few stakeholders may be identified as variable requirements
- Monitoring of requirements along their lifecycle

Requirements may change over their lifecycle from common to variable requirements. So it is important to monitor the requirements, if the classification of the requirements is still valid.

Methods:

Now a range of suitable methods for elicitation will be outlined:

- Interviews

A preferred way of eliciting requirements is to conduct an interview. Such an interview may be executed individually face to face or by phone. It can be held bi- or multilaterally. There are open and formalized interviews.
- Creativity methods (brainstorming, mind mapping, ...)

Especially requirements for new products characterized by a disruptive innovation may be elicited by the use of creativity methods ("Six hats" by DeBono, brainstorming, ...). In the context of product line engineering this method can be applied for identifying completely new aspects of domain engineering.
- Focus Groups

Focus Group is a specific kind of group discussion with a well-defined group of different stakeholders. The composition of the participants with preferably different backgrounds are supposed to leverage existing synergy potential.
- Innovation Workshops

Method to identify innovation potential using creativity methods (brainwriting, ...) in the context of a group meeting
- Customer voice table

Method revealing the veritable requirements behind a rashly expressed customer request. (Part of the Kano model transforming the voice of the customer into inputs for QFD)
- Checklists

Helpful means in order to check completeness of all relevant aspects
- UML

Unified Modelling Language for specification of systems. Following diagrams belong to UML:

 - Use Case Diagrams

Diagram specifying the actor and the use case (action)
 - Class Diagrams

Other diagram of UML representing the static structure of classes (generic term depicting and abstracting common aspects of individual objects)
 - Sequence Diagrams

Interaction diagram representing the exchange of messages between objects

- “Requirements Patterns” for requirements in natural language

Reusable templates for specific types of requirements (e.g. performance, reliability...) leading to a complete set of requirements considering all relevant aspects

- IRENE (see Section 3.1)

IRENE is a method to elicit requirements via the goal model approach. The tree representation illustrates all functional and non-functional requirements with their dependencies and the related prioritizations.

- (contribution of SAP on agile methods in RE)

Analysis:

The main challenges concerning requirements analysis within domain engineering are [Pohl et al. 2005]:

- Consistency Check
see [Lauenroth et al. 08]
- Defining Requirements Variability
Based on the results of the commonality and variability analysis, it is essential to develop a variability model which illustrates:
 - variants
 - variation points
 - variability dependencies
 -

Process Steps:

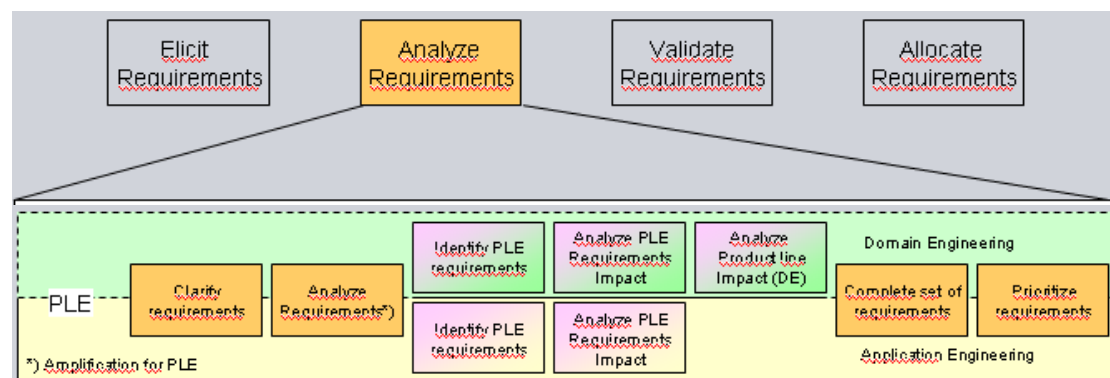


Figure 5: Requirements Analysis Process.

Methods:

- Domain Analysis

e.g. Feature Oriented Domain Analysis (FODA) [Kang et al. 1990]:

FODA has been developed at SEI in 1990. It consists of three phases (Contextual Analysis, Domain Modelling and Architecture Modelling). For requirements engineering only the first two phases are relevant:

- Contextual Analysis covers all questions concerning the scope and borders of the domain and the interfaces to adjacent domains and their objects. Results of this contextual analysis are:
 - Context model
 - Structure model
- Domain Modelling is divided into three specific activities:
 - Feature Analysis
Result of the feature analysis is a feature model represented by a feature list.
 - Information Analysis
The Information analysis identifies the objects of a domain and describes the relations between these objects in an ER diagram.
 - Functional Analysis
The Functional analysis considers the data flow within the domain as well as the state change of the affected objects. That's why two diagrams result from this activity (data flow diagram and state chart diagram).

In [Schleicher2004], FODA is described by the following table:

Phase	Activity	Result	Representation
Contextual Analysis	Contextual Analysis	Context model	Data flow Diagram
		Structure Model	Block Diagram
Domain Modelling	Feature Analysis	Feature Model	AND-OR-Tree
			Feature List
	Information Analysis	Information Model	ER-Diagram
	Functional Analysis	Data flow Model	Data flow diagram
		State chart Model	State chart diagram

Table 1: Feature Oriented Domain Analysis.

The architectural aspects of the third phase (Architecture Modelling) are not considered here, but some of the methods described here are used across several phases.

- Commonality and variability matrix [Mikyeong et al. 2005]

Methods to identify which requirements are mandatory requirements for all variants and which are optional requirements.

The matrix described in [Mikyeong et al. 2005] lists all identified requirements in rows and all considered applications in columns.

An "O" in a cell means that the requirement of this line is relevant for the application of this column; an "X" says that this requirement is not valid for this application.

The CV property denotes, if a requirements is bound for commonality or if it is optional. If the ratio is more than 50%, it is a candidate for "Common" (C), otherwise it is "oPtional" (P):

Req	CV property / Ratio	App1	App 2	App 3	App 4	App 5
Req 1	C / 100%	O	O	O	O	O
Req 2	C / 100%	O	O	O	O	O
Req 3	C / 100%	O	O	O	O	O
Req 4	C / 100%	O	O	O	O	O
Req 5	P / 40%	X	X	X	O	O
Req 6	P / 20%	X	X	X	O	X
Req 7	C / 100%	O	O	O	O	O
Req 8	C / 60%	O	O	X	X	O

- User Centered Requirements Engineering (UCRE) (see chapter 3.5)

User Centered Requirements Engineering describes a further stage of the commonality/variability (CV) matrix of [Mikyeong et al. 2005]. In addition to it, UCRE allocates prioritizations and regards the single stakeholder group prioritizing the particular requirements (see chapter 4.2).

- Decision Modelling

The INDENICA-approach for decision modelling provides a decision making support to the engineer on a conceptual, methodological, and tool-supported level. Details see [INDENICA D1.3.1]

- Feature Modelling:

Feature Modelling is widely used for the representation of functional requirements for product line engineering.

It provides a specific representation of mandatory and optional features and is able to describe several alternatives linked by OR-connection:

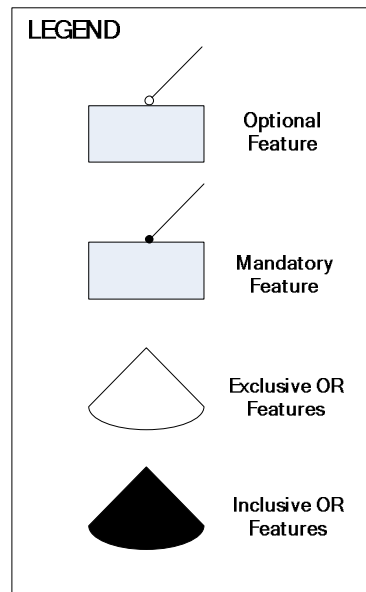


Figure 6: Legend for Feature Modelling Diagrams.

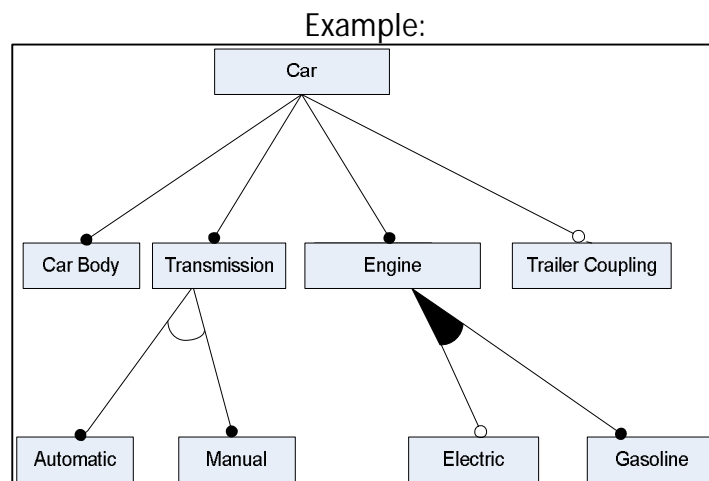


Figure 7: Example for a Feat0075re Model.

Because of these illustration facilities, feature modelling is very well suited for representation of commonality and variability aspects.

- IRENE

Goal model approach for elicitation and analysis of requirements

Support is provided by the tool IRET (see Section 4.1)

- Further methods for requirements analysis

Apart from the methods described above, there are some other methods useful for requirements analysis:

- Scenario Methods

Scenarios describe typical interactions between the users and the system (or between systems). Such scenarios may be used to analyse the desired system behaviour and therefore even to reflect the validity of requirements. In the research work of Colin Potts [Anton, Potts 2009],

scenarios have been postulated for testing the utility and acceptance of the system. Based on the principle of scenarios some methods have been developed:

- Inquiry Cycle developed by Colin Potts is a cycle consisting of three steps in a cycle: Formulating, critical review, refinement. It also uses the goal-oriented approach for requirements analysis. [Potts et al. 1994].
- SCRAM (Scenario Based Requirements Analysis Method):
Alistair Sutcliffe developed this method, which provides a specific modelling language to describe scenarios [Sutcliffe1998].
- GBRAM: Goal Based Requirements Analysis Method
GBRAM is the systematic application of inquiry questions for the analysis and refinement of goals, scenarios and obstacles as well as to attribute them to agents. [Anton, Potts 2009]
- Simulation Methods
The simulation of a system (e.g. mock ups, prototypes etc.) allows to test the “look and feel” or the usability of a system without the need of a fully implementation.
- Scoring / Rating Models
For prioritizing requirements, there exists a wide range of procedures:
 - Clustering / Hierarchy
In a first step, it is necessary to arrange requirements in a structure, which guarantees that only requirements of the same hierarchy level are compared with each other. In the next step typical prioritizing methods like
 - QFD (Quality Function Deployment) and
 - AHP (Analytical Hierarchy Process)
 may be applied.

Documentation:

The main challenges concerning requirements documentation within domain engineering are:

- Documenting variability in requirements artefacts
As well as in general requirements engineering, there are different ways of documenting requirements. In domain engineering the additional challenge lies in the documentation of variability:
 - Natural-language documentation
It is helpful to use graphical elements
 - Documentation by graphical models
 - Describing requirements variability in a feature tree
 - Describing requirements variability in a Use Case model

- Describing requirements variability in other models

Tracing between artefacts and model

For all options it is essential to define and maintain the traceability from variability model to the chosen artefact. Methods:

For documentation of variability in requirements artefacts following methods and tools may be helpful:

- Requirements Engineering Tools (e.g. DOORS, CaliberRM, ...)

When documenting variability in requirements by the use of tools like DOORS or CaliberRM, the database features of these tools bring high benefit. Relations and dependencies can be described and specific views triggered by a filter to get the requirements of a single variant can be selected.
- Use Cases

By use of methods like UML (see "Elicitation") or Feature Modelling (see "Analysis") commonality and variability of requirements can be described.
- "Requirements pattern" for requirements in natural language (see "Elicitation")
- Decision Modelling (see Deliverable 1.3.1, 1.3.2 in INDENICA)

Validation:

The main challenges concerning requirements validation within domain engineering are:

- High quality for common requirements

As common requirements have an immense impact on several or even all applications of a product line, the requirements engineer in domain engineering has to focus especially on the quality of the requirements.

The quality aspects for general requirements engineering are valid to a special degree.

Due to the complexity in domain engineering, it is an even bigger challenge to comply quality attributes as "unambiguous" or "consistent".
- Validation of variants

Depending on the relevance of requirements for specific variants, the specific validation attributes of the requirements have to be set accordingly.

Thus two effects can be noticed:

 - All relevant scenarios are covered by validation -> all variants and their combinations can be validated
 - Unrealistic scenarios are not considered by validation -> no need for use-less excessive validation efforts

Process Steps:

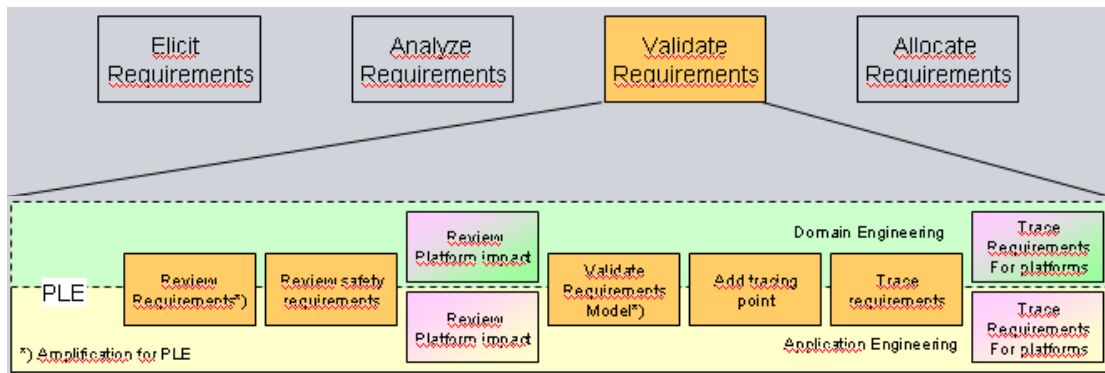


Figure 8: Requirements Validation Process.

Methods:

- All known review methods, like
 - Walkthrough
 - 4 eyes review
 - Formal inspection
 may be applied for validation of commonality and variability aspects.
- Simulation and Scenario methods (see "Analysis")
- Prototypes

Allocation to releases:

Process Steps:

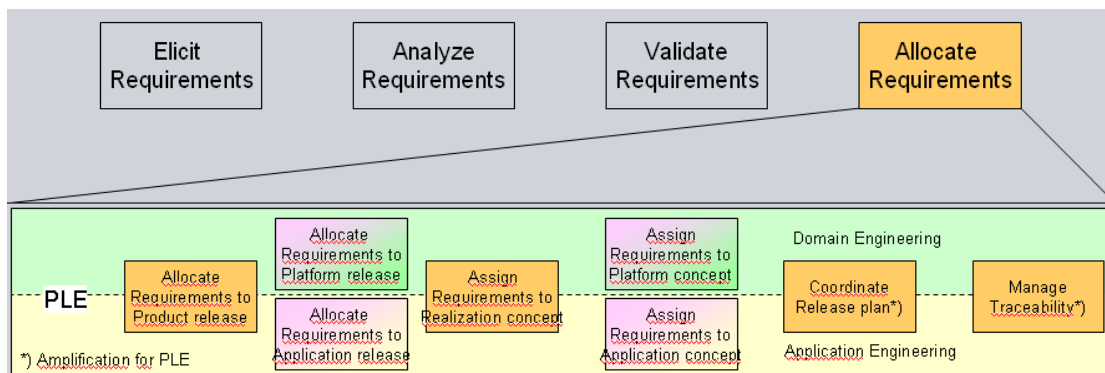


Figure 9: Requirements Allocation Process.

- Allocate requirements to release
 - Allocation to product release
 - Allocation to platform release
 Beyond the allocation of the requirements to specific product releases, in case of product line engineering the determinations of the scoping process (see "Scoping") have to be refined furthermore resulting in an allocation of identified domain requirements to platform releases and application requirements to application releases.
- Assign requirements to realization concept
 - Assignment to product concept

- Assignment to platform concept

For product line engineering, the architectural decision has to consider the scoping determinations and consequently the differentiation between components for the platform and components for specific applications.

- Coordinate release plans
- Manage traceability (see chapter 2.1.1)

The allocation to releases results in a release plan, whereas the “reference architecture” is the outcome of the assignment to realization concepts. Both activities are executed iteratively and parallel.

Methods:

- Recursive Use of QFD (Quality Function Deployment)
Recursive use of QFD delivers an assignment of requirements to realization elements on different architecture levels for platform and applications.
- [ReleasePlanner][®] offers a facility to assign features to releases according to prioritization determined by all involved stakeholders.
- Horizontal and vertical tracing
For product line engineering traceability is a crucial factor.
In addition to the aspects which have to be considered in a product development, tracing for product line engineering has to face following challenges:
 - Tracing between different variants
 - Bi-directional tracing between platform and application artefacts

2.2 RE for Application Engineering

The main characteristic for requirements engineering within application engineering consists in the presence of already existing requirements artefacts arisen from domain engineering. In the INDENICA context this will be the virtual service platform. The applications built by application engineering will be the applications using the services from the platform.

These results serve as essential input for all RE activities within application engineering, as the main target of product line engineering is to reuse as many artefacts built in domain engineering as possible for application engineering.

Thus for requirements engineering that implies the reuse of existing RE artefacts from domain engineering as [Pohl et al. 2005]:

- Common requirements
- Variable requirements

INDENICA Work Package 2 focuses on variability modelling in the specific INDENICA service oriented context.

2.2.1 Requirements Management

Requirements Management for Application Engineering has to ensure the consistence of all requirements of each specific application to

- other requirements of the same application
- the requirements of the core asset base
- the implementations within the applications
- the changes along the product lifecycle

2.2.2 Requirements Development

Elicitation:

The main challenges concerning requirements elicitation within application engineering are [Pohl 2005]:

- Communication of requirements artefacts from domain engineering to the stakeholders
Different from elicitation for single product development, in application engineering it is necessary to inform all relevant stakeholders of the applications about the variety of already existing common requirements and variable requirements assigned to specific variants and the related variability model.
Based on these prerequisites the next activity has to be executed:
- Establishing a set of variants appropriate for specific application
Using now this input from domain engineering, it is necessary to select - beside the mandatory common requirements - appropriate variable requirements from different variants.
After this evaluation a certain set of requirements - common requirements and variable requirements arisen from different variants - is defined which already fulfil a certain percentage of the original requirements of the stakeholders.
The remaining requirements not covered by the selected domain requirements are considered in the next step:
- Establish the delta between domain and application requirement artefacts:
Usually not all requirements of the application can be satisfied by domain requirements. These application specific requirements have to be elicited additionally to the domain requirements and are subject of the subsequent analysis activities.

Analysis:

The main challenges concerning requirements analysis within application engineering are [Pohl 2005]:

Additionally to the analysis activities like "Consistency checking", as described in the chapter of domain engineering, it is essential to regard the relationships between domain and application engineering activities and artefacts. Especially the analysis of deltas between the domain variability model and the application variability model is crucial for the further engagement in requirements analysis:

- Impact analysis for deltas between application requirements to existing variability model w.r.t. existing variation points:

- If for an already existing variation point a specific option for this application is missing, it might be necessary to add a new variant to an existing variation point.
- If an existing variation point does not cover the correct variants, it might be necessary to modify these existing variants
- Impact analysis for deltas between application requirements to existing variability model w.r.t. common requirements:
A common requirement might convert from common to variable, if a new aspect - not yet considered in variability model - has been introduced. So a new variation point has to be added.
- Decision about implementation of deltas
For each identified delta, it has to be decided, if this new part of the variability model will be developed. Here, structuring and prioritizing activities as described in the part "Domain Engineering" might be helpful.

Documentation:

The main challenges concerning requirements documentation within application engineering are [Pohl et al. 2005]:

- Documenting all requirements used from domain engineering:
Either the common requirements mandatory for all applications of this product line either the variable requirements of the domain variability model used for this application have to be described here.
- Delta between domain and application requirements
All requirements which have not been derived from domain variability model (requirements which are new or modified)
- Application variability model:
- Also the variability model of this application with its variants has to be described.

2.3 Relation to Agile Methodologies

Even in larger enterprises we can observe a clear shift away from traditional requirements engineering towards more agile methodologies. A methodology to elicit and analyse requirements that is applied (i.e., inside SAP) is User Story Mapping (USM).

User Story Mapping helps to design products with customer in focus and provides a methodology to establish the backlog that is needed to develop these products. User Story Mapping helps teams get a common understanding of the requirements from the user's point of view and it facilitates the backlog creation. This has been confirmed by the USM pilots at SAP. According to them, the main benefits of USM are:

- Speed up of work of development teams
- Improvement of backlog quality
- Improvement of communication within the team, with other teams, and with customers

Figure 10 gives an overview of the elements of a user story map.

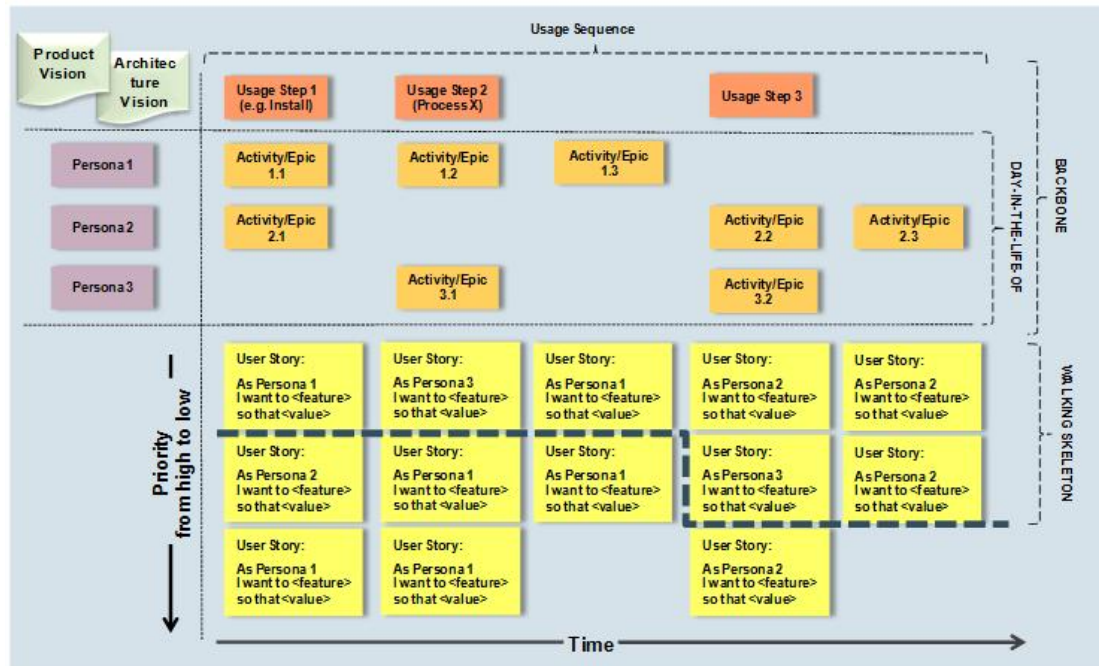


Figure 10: Elements of a User Story Map.

User story mapping typically has the following phases, as also depicted in the figure above:

- **Product vision:** The product owner and the solution owner develop a product vision in close collaboration with the customer.
- **Backbone and architecture vision:** Once the business case is clear, the product owner develops the backbone of the user story map: personas, usage sequence and activities. The architect drafts the architecture vision.
- **User stories:** In a USM workshop with the team and an USM coach, the backbone is further refined: the team defines the user stories thus creating the backlog. Ideally the whole team collaborates to gain a common understanding. This includes the product owner, solution owner, scrum master, architects, developers and information developers.
- **Prioritizations, estimating, slicing:** Once the backlog is created, the team prioritizes the backlog, estimates and prioritizes the user stories and defines the release or sprint backlog.
- **"Deep" backlog:** Typically after the USM workshop, the team reviews the backlog and defines the "deep" backlog for the next sprint and enters it in the backlog management system, e.g. Jira.

The product vision is a short description of the product that is going to be developed. For smaller projects, a few sentences can be enough. For bigger projects a few pages might be necessary. Ideally, the product vision is developed by Solution Management, Product Owners, and Architects in close alignment with customers and stakeholders. The product vision helps the team to understand and agree on the big picture. Typically, the Product Owner presents the product vision in the USM work-

shop. To make sure everybody has a clear understanding of the product vision, one might want to do team exercises in the workshop:

- **Product box:** The team creates the product box for the new product, e.g. what features, benefits etc. The goal is to define a physical item that can be put in a shelf in a shop and which will attract potential customers with appropriate marketing messages.
- **Magazine article:** The team writes a short article, e.g. a text about what would the press like to report about a product?

The architecture vision is a short description about the architectural context of the product, e.g. relevant platform, framework, technology stack etc. If the product already exists, the architecture vision typically also contains boundary conditions and constraints. The architecture vision is developed by the architect and presented in the USM workshop. For the USM workshop, the main aspects of the architecture are enough. There is no need to have a detailed description. The architecture vision might also change during the course of the workshop.

A persona describes an archetypical user of a system. Personas facilitate empathy: they help to understand the requirements of real users and thus make better decisions during the development process. The following Figure 11 depicts some examples:

Name	Candice Clever	Angela Coal	Michael Deal
Picture			
Role	Product Development Analyst	Customer Service Agent	Sales Manager (working at carrier)
Skills	Knows Market very well Interaction with customers and vendors	Trained on the job Permanent interaction with computers	Knows his customers/ partners good at selling Low knowledge of software Knows Market and competitors
Goals	Explore new market opportunities Define products for it	Managed high amount of work Efficient and fast quote/order entry	Successful quote development and contract
Pain Points	Insufficient information about market situation + trends	System blocks entries Too much mouse navigation or scrolling	Gap between contract and real life orders 360° view on customers in one place missing

Figure 11: Example for Personas used in User Story Mapping.

Personas should be based on actual knowledge of real users. For example, it is possible to observe users in their real job environment or one might conduct structured interviews with real users. Then the research results are summarized by defining corresponding personas, each with a name - e.g. Sally Sales - relevant characteristics, motivations, requirements etc. Sometimes even a photo is assigned to make the persona more "real". Typically, the Product Owner develops the personas. However, it's also possible to develop the personas in the USM workshop. Ideally, this should be based on research data. If that's not possible, one should make sure you verify the personas with customers.

The usage sequence shows the high level steps a customer will do with the software. These usage steps are depicted as an end-to-end sequence which usually starts with Install or Setup. The usage sequence does not differentiate between steps of different users. It's rather the end-to-end customer view of the product. The usage sequence is typically provided by the Product Owner. Ideally, the usage sequence is

based on customer research. It's also possible to develop the usage sequence in the workshop based on team discussion. The usage sequence must cover the main steps end-to-end.

Activities - sometimes also called epics - describe typical things a persona wants to do with the product. Activities are developed by breaking down the usage steps into activities for the various personas. Sometimes, a certain usage steps can be broken down into activities for several personas, sometimes only one persona is involved. Typically, the Product Owner prepares the activities / epics as input for the USM workshop. However, they can also be developed in a team discussion in the USM workshop.

A user story describes the needs of users in the following format:

As <persona> I want <need> so that <business goal>

Example: As a Frequent Flier, I want to rebook a trip so that I save time when booking trips that I take frequently.

Each user story is written on an individual card. On the back of each card, the acceptance criteria for this user story are written down: How can it be tested, demoed or verified? During the USM workshop, a rough idea is often enough. It's possible to refine the acceptance criteria shortly before the next development sprint starts. This helps avoiding waste – teams don't want to define detailed acceptance criteria for user stories that will never be implemented. The team develops user stories during the USM workshop. Usually there is a separate brainstorming session used for that.

Once all user stories have been collected, it must be determined which ones should be implemented in the next release. To do so, the moderator of the USM workshop draws a line: all user stories above the line are going to be implemented in the next release. This release scope should at least include the viable scope. An important aspect is this task is the consideration of dependencies: if a user story is a prerequisite for another user story above the line, this user story also needs to be above the line. One might also draw the lines for sprints and future releases. Usually there should not be a plan that is too far ahead. When it comes to the actual prioritization of user stories, the following factors need to be considered: business value, technical risk and effort. For each of these aspects, different methods can be applied. For example, to assess the business value, one might want to consider the Kano model that differentiates between feature types: delighter, satisfier, must-have. To estimate feasibility, one might want to use techniques such as: in/out vote for each story, planning poker or magic planning. However, some teams do the effort estimation later. Then the release line just represents the common gut feeling regarding what is feasible for the next release.

User Story Mapping has been successfully applied to application engineering. Inside SAP there are a lot of product teams which work in an agile mode and which apply USM to define the backlog for their products. However, there are no reported experiences which it comes to eliciting the requirements for a platform. This is due to the blurry picture of the stakeholders of a platform. While it is relatively easy to define personas for an actual application, this task is much harder for a platform.

2.4 Application of process on virtual service platforms

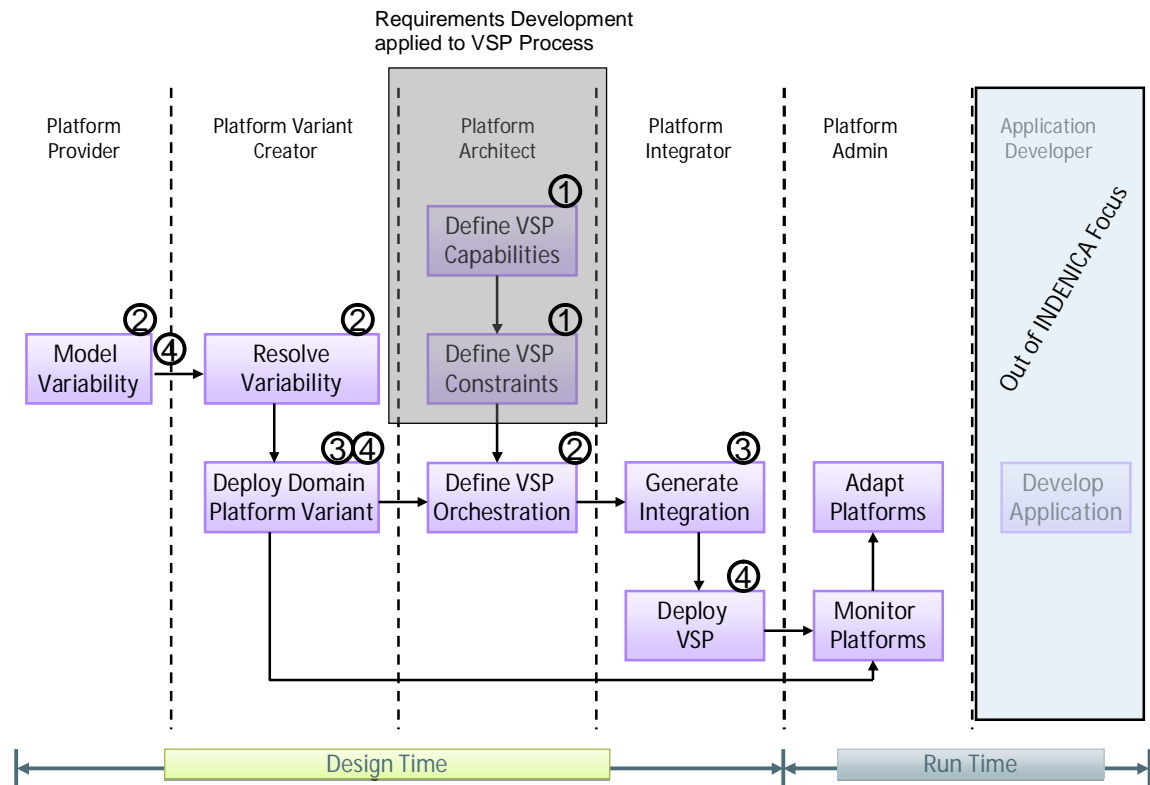


Figure 12: Requirements Development applied to VSP process.

The INDENICA Process as specified in [INDENICA D3.2] has a role oriented process view whereas the generic platform domain and application development process as shown in Figure 1 is phase oriented.

Mapping the four phases: requirements engineering (1), design (2) realisation (4) and testing (4) to the INDENICA process show that the two role related activities “Define VSP Capabilities” and “Define VSP Constraints” have to cover these phases.

INDENICA methods and tools support the definition of VSP capabilities and constraints with following methods and tools:

- ROI calculation for scoping the VSP approach
- IRENE for eliciting and analysing capabilities by modelling goals, entities and operations
- User Centered RE for analyzing and prioritizing goals
- IRENE for validating capabilities with applying formal methods on the goal models

2.5 ROI Calculation

The generic ROI formula presented in [D.1.2.1], chapter 5.1 is usually not applicable in its generic appearance. Too many terms have to be estimated, were most of the terms happen to be estimations of future occurrences.

$$C_{ple}(X, V) = C_{org} + \sum_{v=1}^V \left[\sum_{x=1}^{X(v)} [C_{cab}(v, x) + C_{reuse}(v, x) + C_{specific}(v, x)] \right]$$

Formula 1 : Development cost for a applications based on a core asset base

V	= number of versions
X	= number of applications or product lines, based on the related Core Asset Base
C _{org}	= Cost for reorganization, process improvement, training, etc.
C _{cab}	= Cost for Core Asset Base development (incl. test)
C _{reuse}	= Cost for reusing core assets (e.g. adaptation, configuration)
C _{specific}	= Cost for product line specific development (incl. test)

The accuracy of future estimations can be increased, if the terms can be extrapolated from past behaviour. [D.1.2.1], chapter 5.2 explained how this was done in a customer project. The basic assumptions for the business adapted formula and tool presented in this chapter are:

- (1) Costs are dominated by development efforts
- (2) Linear dependency between development cost and test cost
- (3) Linear dependency between the costs from one version to next version
- (4) Constant number of product lines and linear effort dependency between them.

Assumption (1) heavily depends on the business case. [D.1.2.1], chapter 5.3 explains how additional cost terms have to be considered if (1) is not true. Assumption (2) is likely true for most development projects. Assumption (3) depends on the developed product and the maturity of the core asset base. Non-linear dependencies may show up in certain cases and an in-depth analysis of the current situation is necessary before it is allowed to use assumption (3); e.g. C_{reuse} may increase from version to version due to increasing management and retrieval costs for a growing core asset base. If non-linear dependencies show up, then the tool needs to be adapted accordingly. Assumption (4) depends on the organization's situation.

Using these assumptions formula 3.2 translates to:

$$C_{ple}(X, V) = C_{org} + [C_{cab} + X \times (C_{reuse} + C_{specific})] \\ + (V - 1) \times [C_{cab} \times F_{cab} + X \times (C_{reuse} \times F_{cab} + C_{specific} \times FS_{adapt})]$$

This can be resorted to:

$$C_{ple}(X, V) = C_{org} \\ + (C_{cab} + X \times C_{reuse}) \times [1 + (V - 1) \times F_{cab}] \\ + X \times C_{specific} \times [1 + (V - 1) \times FS_{adapt}]$$

and with

$$C_{org} = N_{empl} \times C_{empl} / W_y$$

$$C_{cab} = C_{prod} \times F_{reuse} \times F_{overhead} + C_{scoping}$$

$$C_{reuse} = C_{cab} \times FR_{adapt}$$

$$C_{specific} = C_{prod} \times (1 - F_{reuse})$$

The formula can be translated to:

$$C_{ple}(X, V) = N_{empl} \times C_{empl} / W_y \\ + (C_{prod} \times F_{reuse} \times F_{overhead} + C_{scoping}) \times (1 + X \times FR_{adapt}) \times [1 + (V - 1) \times F_{cab}] \\ + X \times C_{prod} \times (1 - F_{reuse}) \times [1 + (V - 1) \times FS_{adapt}]$$

Formula 2: Business case adapted development cost formula

See Figure 2.9 and 2.10 for an explanation of the terms.

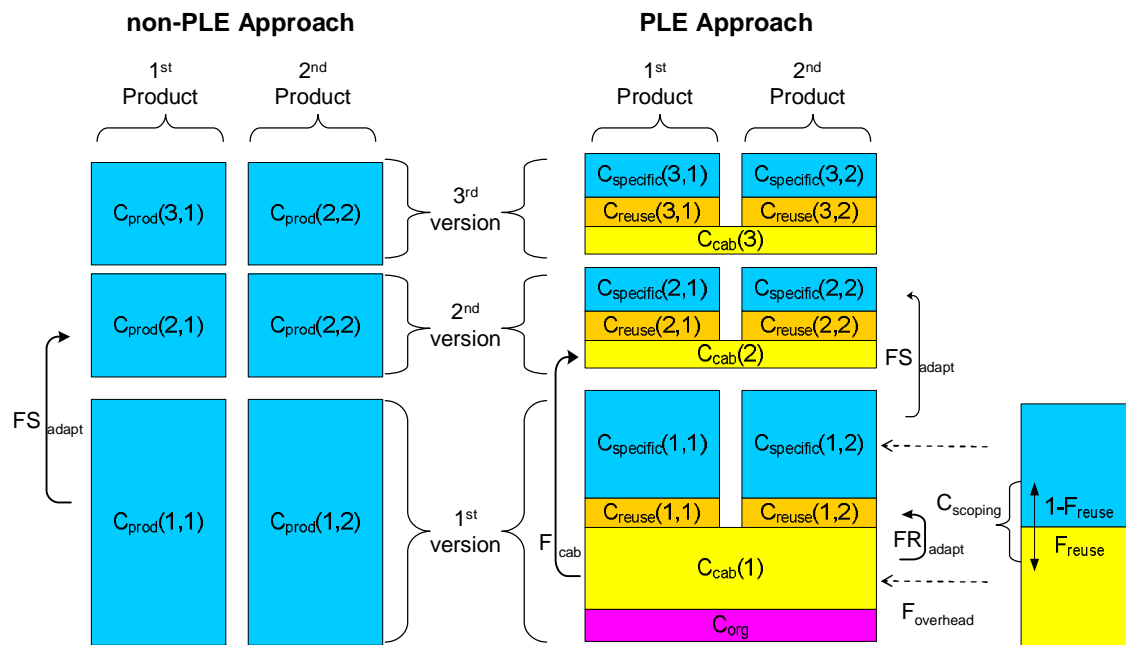


Figure 13: Example cost factors for 3 versions of 2 product lines, CAB cost evenly distributed.

Based on this example a small Excel-Tool was developed that allows a simple application of this business case adapted formula. This tool will be described in this chapter. It should be simple to adapt the tool to other business situations.

Term		Remark
N_{empl}	Number of employees	<i>known figure; organization dependent</i>
W_y	working weeks per year	<i>known figure; country dependent</i>
C_{empl}	Cost (person weeks) per employee for reorganization, training, etc.	<i>estimated</i>
C_{org}	Cost (person years) of changing the organization: $N_{empl} \times C_{empl} / W_y$	<i>calculated</i>
C_{prod_D}	Cost (person years) of developing one product from scratch (non-PLE approach)	<i>known figure; based on past projects</i>
C_{prod_T}	Cost (person years) of testing one product from scratch (non-PLE approach)	<i>known figure; based on past projects</i>
C_{prod}	Cost (person years) of developing and testing one product from scratch: $C_{prod_D} + C_{prod_T}$	<i>known or calculated</i>
FS_{adapt}	Factor for the effort to evolve specific SW for reuse in a product: (=1 for new SW)	<i>known figure; based on past projects</i>
C_{prod_evo}	Cost of evolving one product (non-PLE approach): $FS_{adapt} \times C_{prod}$	<i>calculated</i>
$C_{scoping}$	Cost (person years) for identification of common SW	<i>estimated</i>
F_{reuse}	Reuse Factor; i.e. percentage of reusable SW	<i>estimated</i>
$F_{overhead}$	Overhead Factor for the extra effort to make SW reusable; incl. additional design and testing, documentation, training, etc.	<i>estimated</i>
C_{cab}	Cost of Core Asset Base: $F_{overhead} \times F_{reuse} \times C_{prod} + C_{scoping}$	<i>calculated</i>
F_{cab}	Fraction of Core Asset Base that changes with each new version	<i>estimated</i>
C_{cab_evo}	Cost of evolving the Core Asset Base for the next version: $F_{cab} \times C_{cab}$	<i>calculated</i>
FR_{adapt}	Factor for the effort to adapt core assets for reuse in a product	<i>estimated</i>
C_{reuse}	Cost of using the core assets to build the first product: $FR_{adapt} \times F_{reuse} \times C_{prod}$	<i>calculated</i>
C_{reuse_evo}	Cost of using the core assets from 2nd version: $FR_{adapt} \times F_{cab} \times F_{reuse} \times C_{prod}$	<i>calculated</i>
$C_{specific}$	Cost for initial development of unique SW parts: $(1-F_{reuse}) \times C_{prod}$	<i>calculated</i>
$C_{specific_evo}$	Cost for evolving unique SW parts to build a new version: $FS_{adapt} \times (1-F_{reuse}) \times C_{prod}$	<i>calculated</i>

Figure 14: Example cost factors for 3 versions of 2 product lines, CAB cost evenly distributed.

Figure 14 shows the table of the sheet, where the input figures of the formula can be entered. Only a few figures need to be estimated, some should be known by the organization from experience.

All other sheets use these figures to calculate the cost ramp up in 4 different scenarios:

- (1) green field, 1st version
- (2) green field, version development
- (3) reengineering, 1st version
- (4) reengineering, version development

Scenario (1) compares a non-PLE approach with a PLE approach and evaluates how many Product Lines are needed to reach a cost break even already with the first version. In the example, Figure 15 the cost break even will be reached with 2 product lines based on a core asset base (CAB). With 3 product lines based on a CAB there is already a significant cost advantage compared to 3 independently developed products.

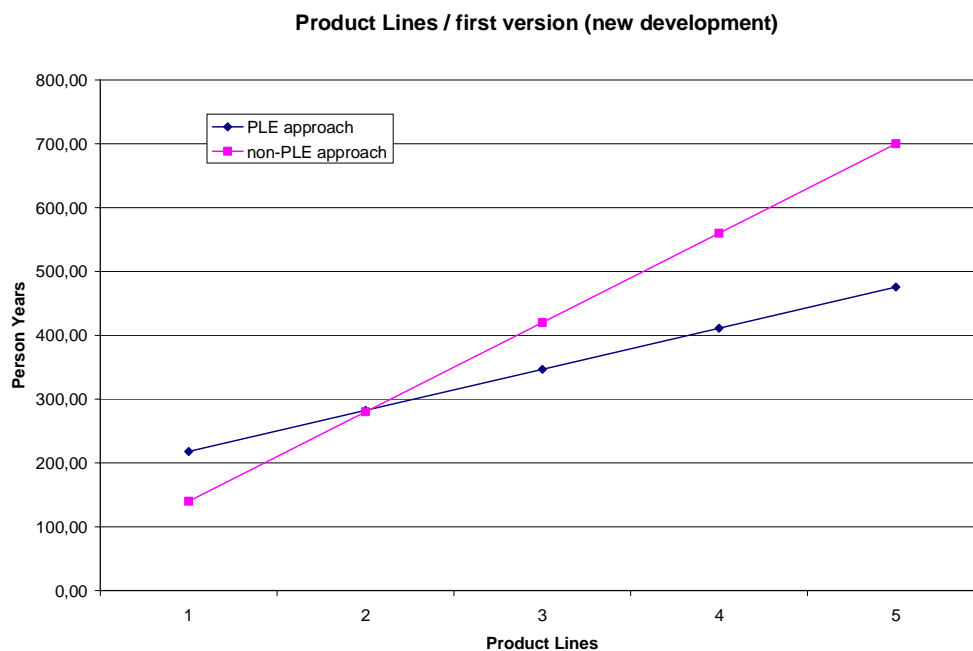


Figure 15: Example for scenario (1); cost break even with 2 Product Lines.

Scenario (2) takes a given set of product lines and evaluates how many versions are necessary to reach a cost break even. The example in Figure 16 uses the same input as the example in Figure 15.

Figure showed already a cost advantage for 3 product lines based on a CAB and Figure 16 shows that this advantage increases with each additional version developed; i.e. over time.

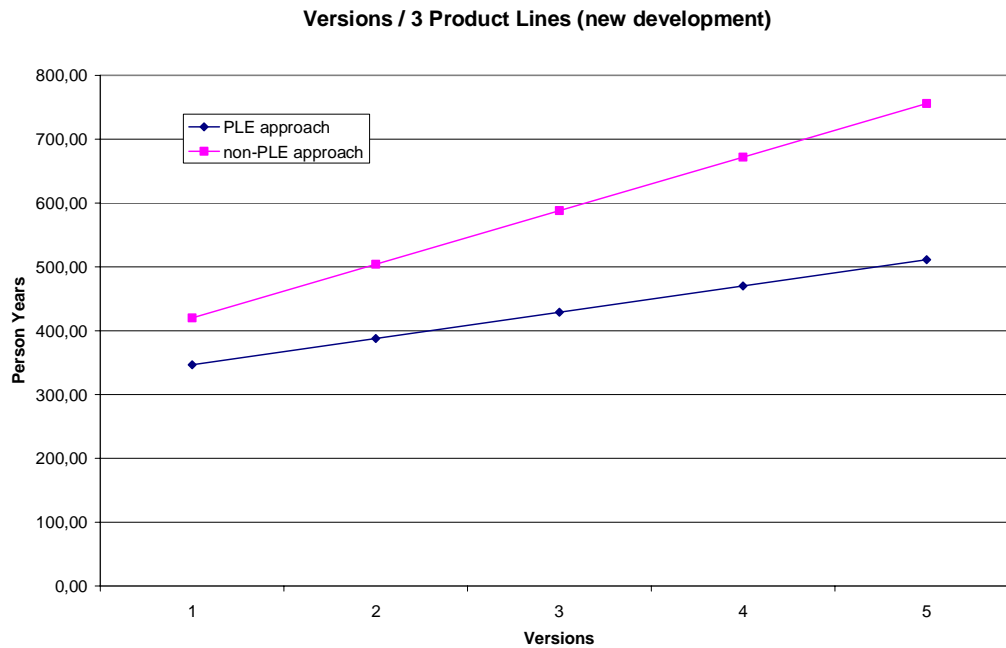


Figure 16: Example for scenario (2); advantage for PLE approach increases with each version.

Scenario (3) assumes that existing products shall be reengineered towards a PLE approach, and evaluates how many Product Lines are needed to reach a cost break even already with the first version.

Figure 16 shows that, again with the same inputs, there is no chance to reach a cost advantage with a PLE approach in the first version. This is expectable because in a reengineering scenario the non-PLP approach does not show the large development costs for the first version, whereas the reengineering cost for a transfer from the current non-PLP approach to a PLE approach needs significant efforts for the first version, which are (in this example) assumed to be as high as for a development from scratch.

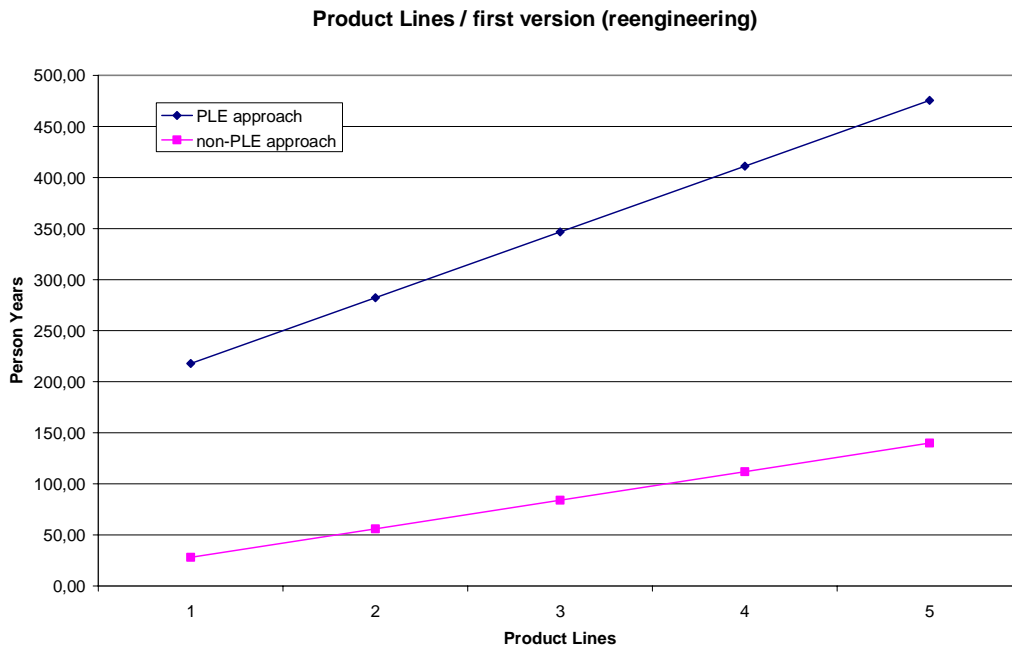


Figure 17: Example for scenario (3); no cost break even with first version possible.

Scenario (2) showed that a PLE approach gained advantage with each version developed. Therefore scenario (4) again assumes that existing products shall be reengineered towards a PLE approach, and evaluates, with a given set of Product Lines, if a cost break even can be reached with an increasing number of versions.

With again the same input values as in the other examples, here a cost break even for 3 product lines based on a CAB shows up with the 7th version; i.e. an income break even may show up with version 13 or 14. This is far in the future and many disturbing events may show up in-between that influence the costs. In this example it would not be advisable to perform a reengineering towards a PLE approach.

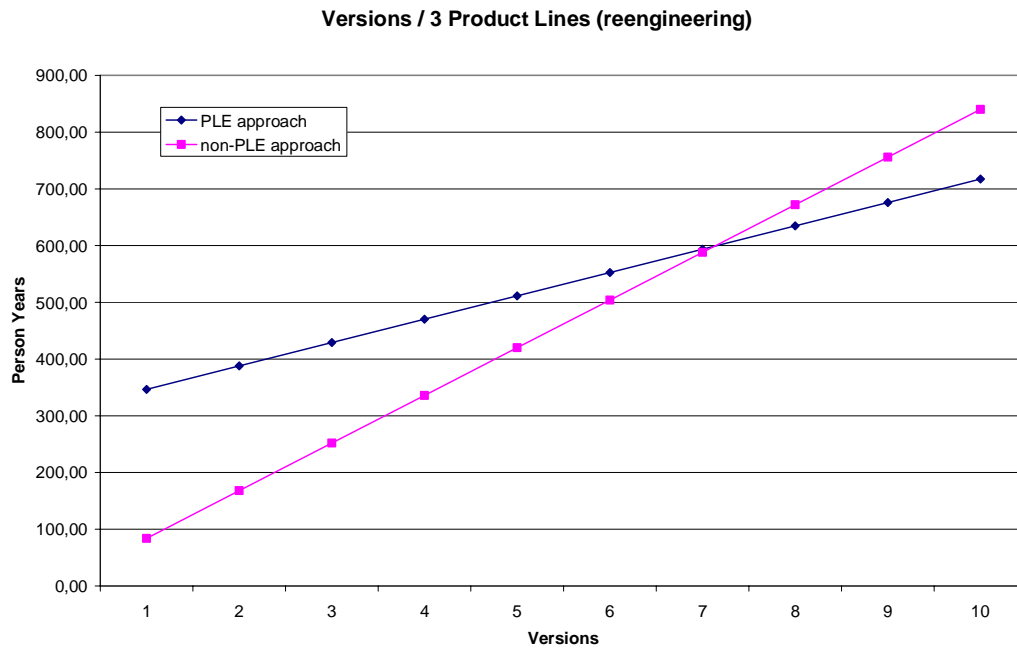


Figure 2: Example for scenario (4); cost break will be reached very late; return on invest questionable.

Tool Support

The Excel spreadsheet for RoI calculation is available under:

<https://repository.sse.uni-hildesheim.de/svn/Indenica/deliverables/wp1/d122-release>

Summary

Depending on the scenario in which an organization operates, the adaptation of the generic Formula 2 in conjunction with a simple Excel application allows a fast estimation if it is recommendable to use a PLE versus a non-PLE approach or if it is advisable to switch over from a non-PLE to a PLE approach. The accuracy is sufficient for strategic decisions and the adaptation reduces the set of terms to be estimated to a handful of values.

3 INDENICA specific notations

3.1 IRENE

IRENE has evolved over the last year. It evolved as planned to both incorporate the new elements foreseen during the project, and also to take into account the suggestions and new requirements from the users. This means that IRENE is now a more complete notation: besides the purely syntactical elements, now it also provides means to ease the integration with other elements in the project, and also to support the actual and effective integration of different requirements models, or different views on the same model.

The next sections provide a brief overview of the main changes and addition to the notation, while Section 4.1 explains the evolution of IRET, that is, the toolset developed in the project to offer IRENE to its users.

3.2 Views

Traditionally, requirements models ---rendered as goals--- are presented as single diagrams. In IRENE, they would be single trees¹, but real problems usually lead to complex diagrams that the user cannot easily manage and navigate. The problem here is not to extend the concrete syntax of the language, but to offer users means to slice their diagrams and identify proper *views* on them. The problem, and then the obvious requirement, became evident as soon as the final users started modeling their problems with IRENE/IRET. Clearly, they needed a better means to:

- Structure models. As said before, when models are not trivial, one must find suitable ways to render them, and to let the user work on them. Views should offer the opportunity to split a complex model into usable and understandable chunks, but it should also allow one to emphasize a particular set of elements. These means that some elements may belong to different views, while others could be rendered in a single view (obviously, each element must be present in at least one view)
- Integrate different viewpoints. Since IRENE was created with the idea of defining a goal-oriented requirements elicitation notation for service platforms, the problem we had to face was the definition of means to start from different viewpoints: the viewpoints of different stakeholders on the same platform or the elicitation of the requirements of different applications that could be based on the same platform. Clearly, one may start from different models and then try to produce a new one by “integrating them”, but this would not be the best to stress the cohesion among these different, but related, models. This is why we think that views can also help here. A view can easily render a particular viewpoint on the same model and then become a way to properly relate a set of (loosely) related IRENE models.

¹ Even if this is not completely correct since dependencies might turn trees into acyclic graphs.

Given these requirements, we borrowed from the way many UML tools work. We decided to distinguish between model elements and representations. Each model element has a single identifier, but this is not new, and now it can be associated with different representations in different views. Each element can only be represented once in one view, but then the actual structure of the different views is up to the user. Whenever a new model is created, it is implicitly associated with a view; the user can always create others as soon as needed.

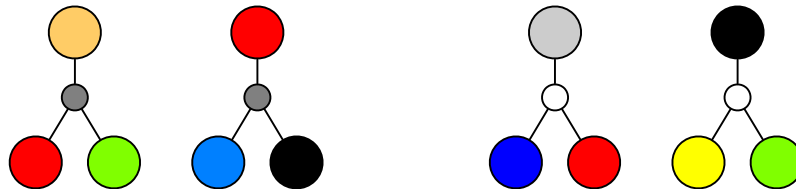


Figure 18: Independent and dependent views.

Given the requirements above, we can have both *independent* and *dependent* views. The former (Figure 18.a) do not share elements, and could be used to unify different trees, that is, to merge the requirements that come from different applications that should be supported by the same platform. The latter (Figure 18.b) share elements, and could be used to structure a single tree into meaningful and usable sub-trees, where the root of a view is a leaf in a previous one.

3.3 Variability support

As already described in Deliverable D1.3.1, IRENE explicitly captures base information for variability decisions and constraints among them. Each element of IRENE can be associated with variability-related comments. The hypothesis is that the user specifies:

- The variable aspects of the element. This information is mandatory to start understanding the actual variability the user wants to embed in the model. If one started identifying variability here, the whole variability specification process would move towards continuous refinement in an incremental way;
- The values varying aspects are expected to assume. Again, this is important to provide sufficient information to enable the actual design of the variability that the user wants to embed in the platform-to-be;
- When this variability should be resolved. Even if, traditionally most of the variability is/was resolved at design time, the particular context, that is, service platforms, may also require a more dynamic approach towards variability, and thus one may say that the actual decisions are taken at deployment-time or even at run-time;
- The links and constraints with other variability requirements. Even if variability requirements are stated independently for each notation element, the user must be aware of links with other elements and possible constraints. Informal comments are just a first step towards the definition of a complete, coherent, and sound variability model of the whole platform-to-be.
- Finally, we also envision the possibility of adding free notes and comments as further suggestions and placeholders for the next steps.

This information is provided mainly as comments, but the user can also exploit a tighter integration with WP2 and exploits the key elements of the Indenica Variability Modeling editor. The purpose of this information is twofold: it feeds the actual design of the variability that belongs to the platform-to-be and provides the basis to initiate the decision process.

3.4 Model integration

One of the key tools needed for modelling the requirements of service platforms is the capability of integrating different views (models). As already said, we think that the requirements may come from different stakeholders, and this does not only apply to service platforms, but they may also come from considering the different applications that are supposed to exploit the services provided by the platform.

After introducing the views, the new version of IRENE also proposes a simple, but effective way for integrating different views. The algorithm is based on some key assumptions:

- The final decision about the actual similarity between two goals, entities, or operations is always up to the user. The automated solution is only supposed to help highlight the similarities, but it does not decide anything.
- In order to keep things simple, the similarity is only evaluated on the names of the different entities. The only exception is that we also consider the priorities associated with goals. This is because priorities are important, and different stakeholders may have different opinions, but if one considered all the properties, the decision process would become too complex and too time consuming.
- The user is in charge of setting the actual “precision” of the analysis. Clearly, the extreme that any name is similar to any other would be useless, but also the strict equality between the two strings could be too tough. This is why we decided to reuse a well-known information-retrieval algorithm to state the similarity between words, and let the user set the filter: from 1, which is the perfect match, to 0, which would mean no match and thus all elements are similar to all the others.
- Finally, to ease the integration and identify a clear starting point, we decided to work incrementally. The user must decide a reference view, which is considered to be the master model, and the algorithm reasons on the similarities, and differences, between this view and the others.

The key idea of the algorithm is simple and neat:

- If entities and actors are not considered to be similar, they are added to the reference views.
- Since goals are organized in hierarchies, and the ones of the reference view must be kept, considered-to-be similar nodes are merged, while those that are not similar are added accordingly by following the patterns of Figure 19.

The figure uses colours to highlight the similarity, or difference, between nodes. All patterns are self-explanatory. The last two rules must add a “placeholder” node to

keep the hierarchy. This means that the proposed result is not complete, but the intervention of the user is required to either give a meaning to the placeholder, maybe by merging it with another goal, or remove it and thus change the topology accordingly.

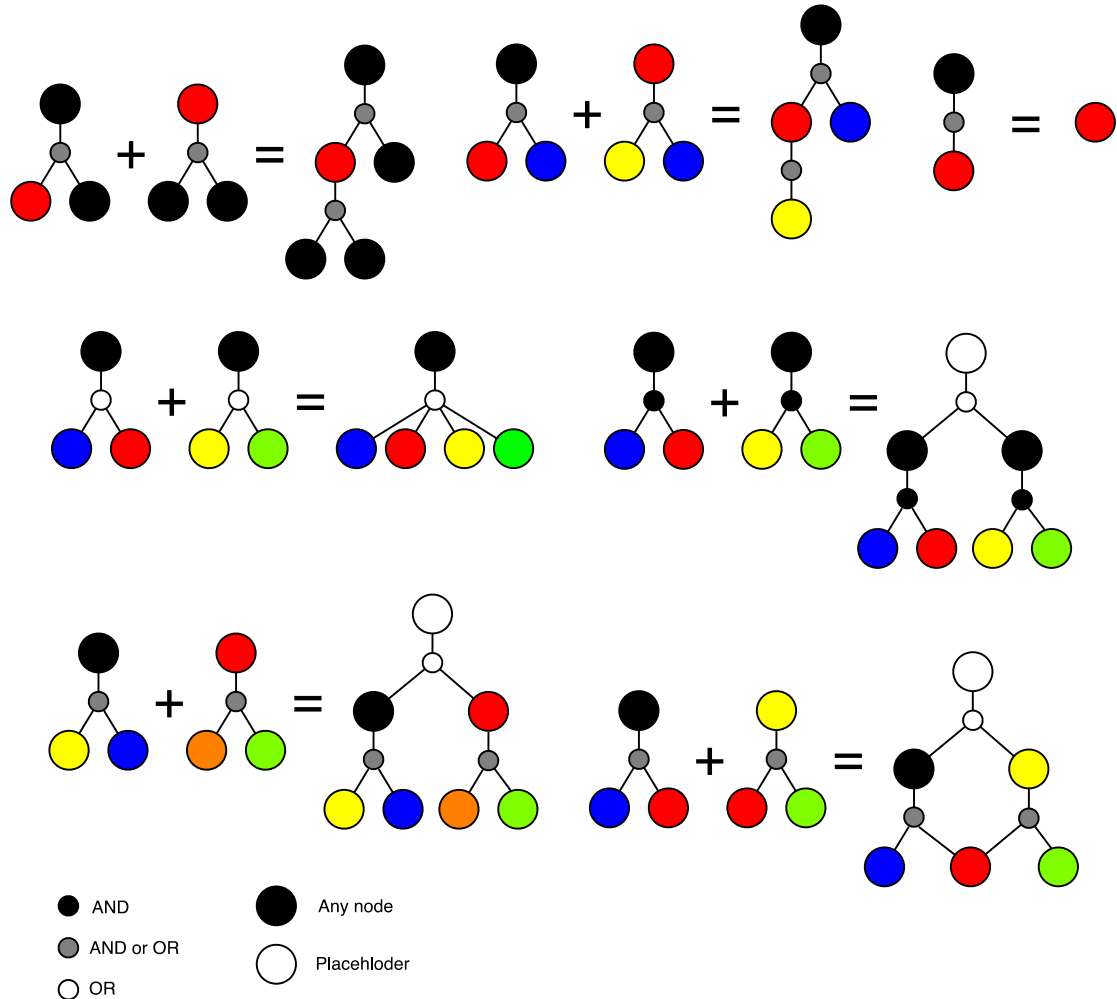


Figure 19: Some of the patterns used to integrate/merge views.

3.4.1 Semantic similarity

The similarity between two words $w1$ and $w2$ is computed in three steps: tokenization, stemming, and distance evaluation. Tokenization decomposes $w1$ and $w2$ in two sets of terms: $w1 = \{t1,i\}$ and $w2 = \{t2,i\}$. It considers case changes, underscores, hyphenations, and numbers. The terms resulting from the tokenization are stemmed: words like sending or exchanged are transformed into their stemmed version: send and exchange. The stemming process is a well-known process adopted by several information retrieval approaches [Lennon et al. 1988]. The third step is about the computation of the similarity $wSim$, obtained by exploiting the assignment problem in bipartite graphs:

$$wSim(w_1, w_2) = \frac{1}{|t_{1,i}|} \cdot maxSim(termSim, t_{1,i}, t_{2,i})$$

Before discussing how *wSim* works, we introduce the bipartite graph assignment problem since it provides the basis for our similarity function. Given a graph $G = (V, E)$, where V is the set of vertexes and E the set of edges, $M \subseteq E$ is a matching on G iff no two edges in M share a common vertex. If M covers all the nodes of the graph, G is bipartite. This also means that each node of the graph has an incident edge in M . Let us suppose that the set of vertices are partitioned in two sets Q and P , and that the edges of the graph are weighted according to function $f : (Q, P) \rightarrow [0..1]$. Function $maxSim : (f, Q, P) \rightarrow [0..1]$ returns the maximum weighted assignment, that is, an assignment such that the average of the weights of the edges is maximum.

The inputs are the two sets of tokens $\{t_{1,i}\}$ and $\{t_{2,i}\}$ that compose the two words to be compared, and function $termSim : (term, term) \rightarrow [0..1]$ that returns the similarity of two tokens. This way, *wSim* returns the word similarity as the sum of the similarities between the pairs of tokens that maximize such a sum. On this basis, *termSim* holds a central role in the computation of the similarity.

The literature proposes several approaches to state the similarity and relatedness between terms. These algorithms usually compute such a similarity by relying on the relationships among terms defined in a reference ontology (e.g., is-a, part-of, attribute-of). Our approach computes the similarity between terms by adopting the solution proposed by Seco et al. [Seco et al. 2004]: they rely on the assumption that concepts with many hyponyms convey less information than concepts that have less hyponyms or any at all (i.e., they are leaves in the ontology).

Note that *wSim* returns the maximum sum divided by the number of terms composing w_1 . Indeed, in the application of the assignment problem in bipartite graphs to our context, set t_1 represents a query, whereas t_2 is what we compare against the query to evaluate the similarity.

3.4.2 Requirements matching

The measure of similarity between operations can also be used to try to match the requirements defined for a new platform-to-be against a set of already existing solutions. The solution is not perfect, but it could be really effective.

The two key ingredients are the IRENE model of a new platform and the services provided by existing solutions (models as actors and operations). This means that either these platforms had been already specified using IRENE, or some “spurious” models have been defined to enable the comparison.

After setting the actual degree of similarity, the same algorithm defined to integrate different views can now be used to simply highlight the similarities, and thus to let the user easily understand what already exists ---or may exist--- and what should be implemented from scratch. The only difference is that in this case, the result is not an integrated view, but the actors and operations of the IRENE model of the plat-

form-to-be are properly annotated to clearly mark those that have already been implemented and those that do not exist among existing platforms.

3.5 User Centered Requirements Engineering

3.5.1 Motivation: Summary of work presented in D1.2.1

In the first version of this deliverable we introduced the aspects of User Centered Requirement Engineering. Based on the ideas of the Johari Window, that identifies four distinct areas of perception, we defined a method for bringing together the viewpoints of a variety of user (or stakeholders) on a system or a platform.

We defined User Centered Requirements Engineering as a well defined part of User Centered Design. For this sake we derived three dimensions that play a role in eliciting and analysing the users' needs:

1. The user dimension describes how close the user is to the system and its user interface(s).
2. The importance dimension describes how important a requirement is for a user or a user group.
3. The requirements dimension files all requirements and the topics that could cluster the requirements to larger logical areas.

3.5.2 Graphical display of Priorities

For the graphical analysis of requirements, user groups and the priorities a three dimensional display could be used like shown in Figure 20.

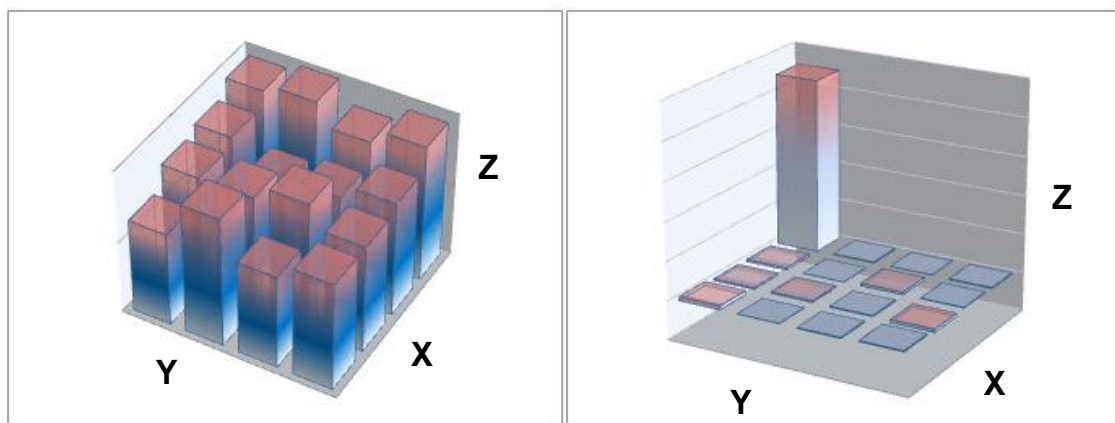


Figure 20: Three-Dimensional Display of Users, Requirements and Priorities

For every user (x-axis) each requirement (y-axis) is tagged with a priority (z-axis). This leads to a multi-column chart that might give an overview on the homogeneity of priorities among users and for distinct requirements. But for a real analysis it has some drawbacks:

- Low priorities could be hidden / invisible

- In case of large set of requirements the requirements axis is much longer than the axis of users

Further details of priority analysis cannot be displayed without display conflicts. Even advanced graphical facilities like changing viewpoints, zooming, selecting did not really provide necessary insights.

For these reasons we decided to go back to two two-dimensional displays for implementing a supporting tool.

3.5.3 Analysis of Conflicting Sets of Prioritization

The graphical analysis of sets of prioritization is done in two views:

The requirements view looks at the variation of priority votes from different users. There are different calculation methods to derive a best-fitting value from a set of estimations:

- The average mean
- The median

For a small set of discrete values the median value is more meaningful and less sensitive with outliers. It is defined as the value that is in the middle of a sorted list of values.

Neither the average nor the media give information on the distribution of values. For this purpose we use the variance that is defined as the mean square deviation from the average value.

The application of these values will be discussed in following chapter, where we can see the implementation in a tool and examples from real data.

Figure 21 shows an example of a user voting on a requirement. Four out of five users voted 1 (unimportant), one voted 4 (very important). The reasons behind that can be manifold and need to be investigated:

- User 4 could have misunderstood the requirement
- All other users could have misunderstood the requirement
- User 4 has a very different view on the platform and thus his voting differs from the others

In any case a discussion, a clarification or further investigation is needed in order to agree on a final priority of the goal.

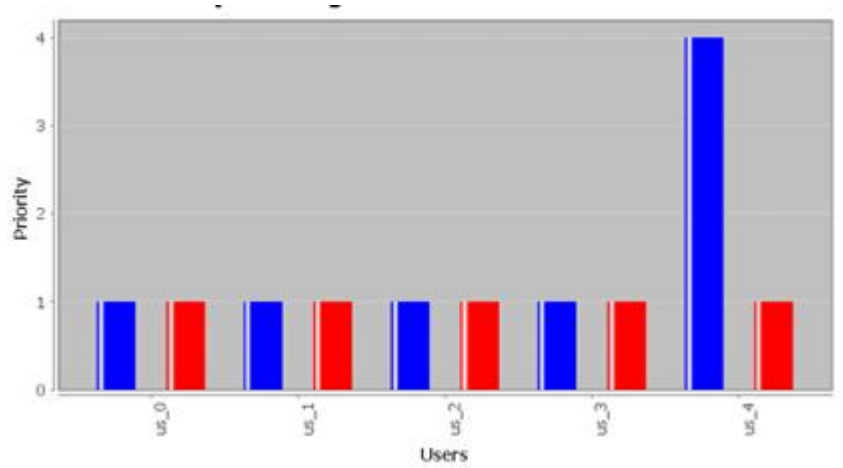


Figure 21: User voting (blue) and median (red) on a single requirement

The user view looks at the voting of a single user. An average or median value is not of interest here, but we can also look at the variance in order to see if the user applied the full range of voting possibilities or a very narrow one.

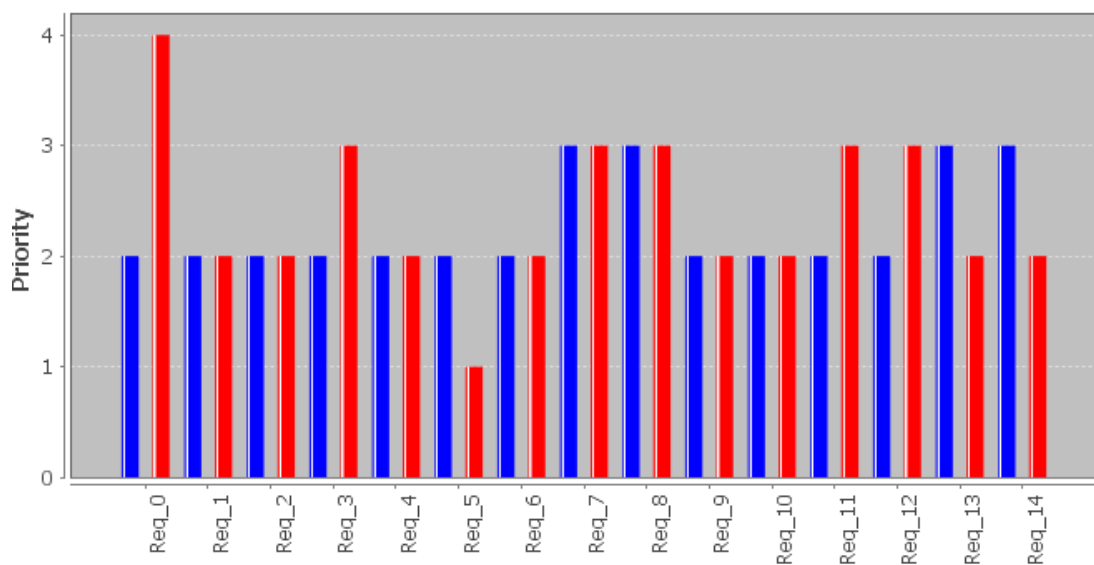


Figure 22: Voting of a single user on all requirements (blue) compared to the median (red)

The user voting in Figure 22 shows that the user never used Priority 1 and 4. This could be an indicator that the user is often quite unsure about the requirements, maybe did not understand the overall context. In such a case a clarification with the user could lead to a new set of priorities that reflect better the user's view on the system.

In order to support prioritization of goals that were modelled with IRET a tool support was implemented. It consists of two parts:

- The goal extractor
- The priority analyzer

Strategy for extracting items from a goal model:

Prioritization Model

...

Workflow

The workflow is shown in Figure 23, the tools and their interfaces are described in chapter 4.2.

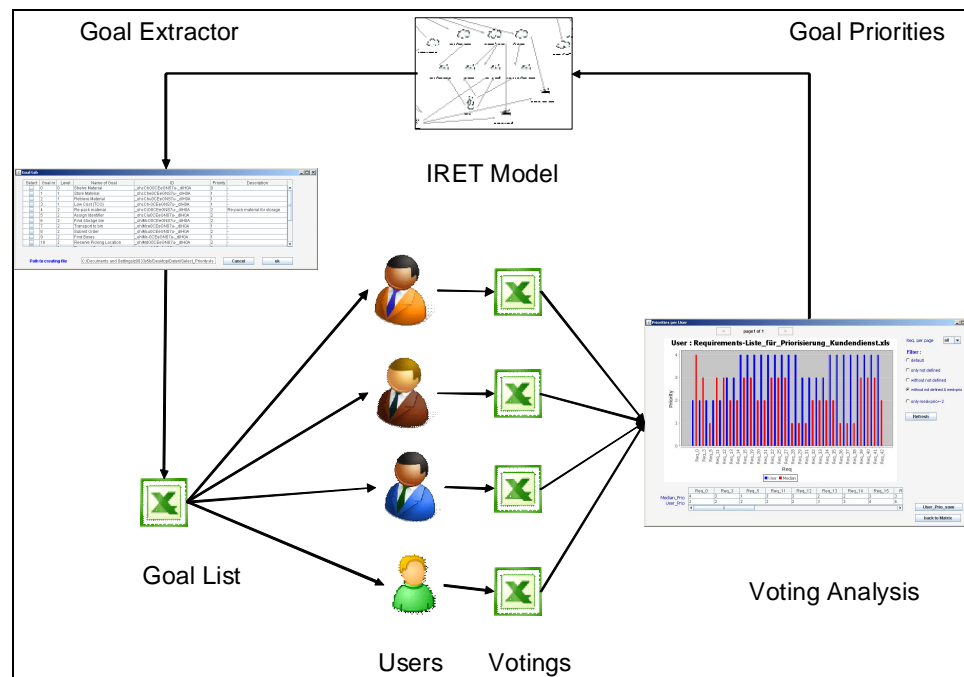


Figure 23: Workflow of Goal Model Prioritization.

4 Tools

4.1 IRET

This section presents the final version of the tool for the goal-based requirement engineering modelling, named IRET (IRENE Toolset). IRET is developed following a model-driven approach based on an EMF model (see [EMF] for details). In [INDENICA D1.2.1] we described the original EMF model behind the first version of the tool. During this second year we finalized the development the tool adding several features. Some of them require an update to the original EMF model. Figure 24 is the object model showing the updated EMF schema behind IRET.

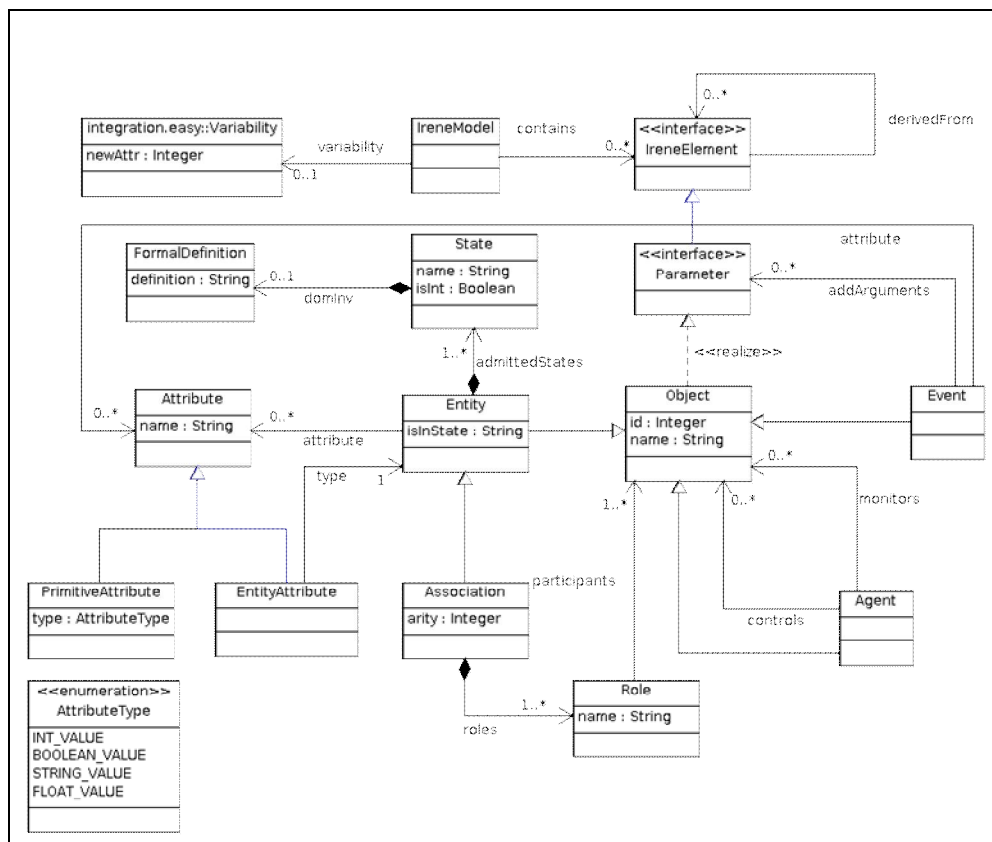


Figure 24: Updated IRENE object model.

In the following subsections we will describe in detail the improvements of IRET, here we briefly introduce the main changes in the EMF model:

- Events attributes: the relation between the Event class and the Attribute class have a 0..* cardinality. This allows the user to describe an event in term some parameter e.g. date time, kind of event, recurrence, ...
- Attribute types can be model entities. the Attributes can be "Primitives" or "EntityAttributes". An EntityAttribute type is defined by another entity of the model. Thus, the modeler can define new Entities and use them as types for other entities attributes.

- The concept of IreneModel has been explicated and a general concept of IreneElement has been introduced. As an example, this allows the modeler to define two models (or one model with two views) reusing the same element in both.
- An IRENE element can be generated starting by another element. An DerivedFrom relation is introduced for this purpose.
- An extension to support the variability model defined by WP2 has been introduced.

The rest of the section is structured as follows: in Section 4.1.1 we explain the extension for managing Events and Entities; in Section 4.1.2 we introduce the multi-view mechanism to manage in several diagrams the same model. Section 4.1.3 presents the model merge process, while Section 4.1.4 lists other minor improvements available in the last version of IRET. Finally in Section 4.1.5 we describe the integration of IRET with other INDENICA tools.

4.1.1 Improvements on Attributes, Events and Entities

IRET implements Entities; as the name suggests, entities represent both abstract and concrete concepts in the real world. An entity has a state and a set of attributes; an attribute is a <key-value> pair describing one property of the entity. It means that in the model the Attribute object is a class with two properties: the name of the attribute (the key) and the type of the value.

The previous version of IRET, described in D1.2.1, implemented a simplified version of the attribute class (Figure – Before): the attribute type could be one of the primitive ones: Number, Boolean and String.

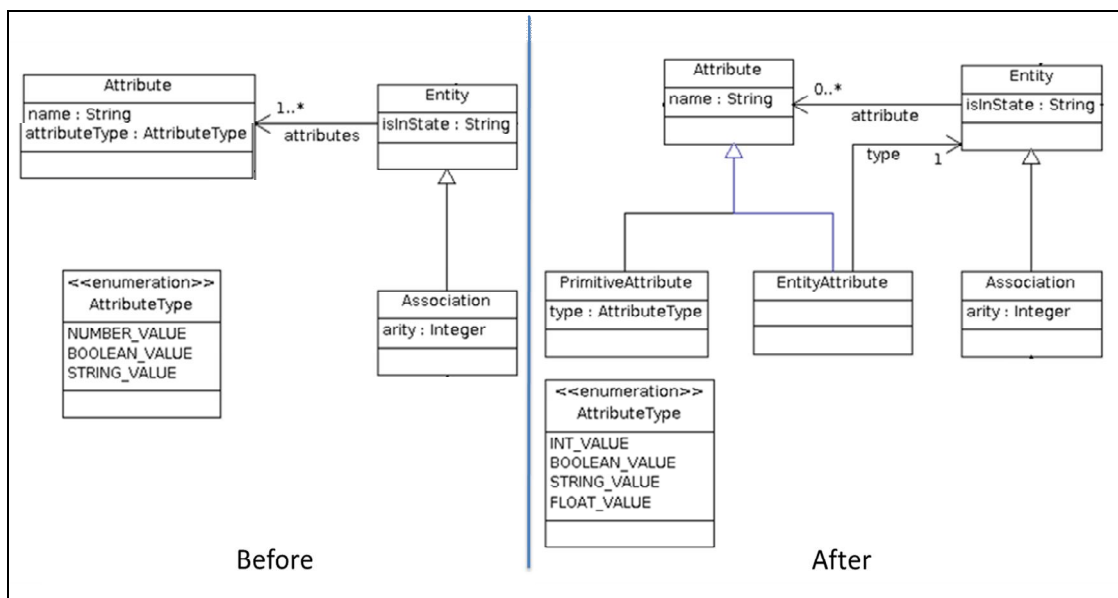


Figure 25: Updated attributes in the IRENE model.

In the new version of IRET we updated this model introducing the idea that attributes can be also of complex types. As shown in the right side of Figure 25 (After), the Attribute class now has two subclasses: PrimitiveAttribute and EntityAttribute. While PrimitiveAttribute models the attribute with value of simple types (Integer, Boolean,

String or Float), EntityAttributes can be used to indicate that an attribute has a complex type (another Entity).

Both Entity and Event have a set of instances of the Attribute class. By a graphical point of view, users can create and delete attributes through a dedicated tab in the Property View.

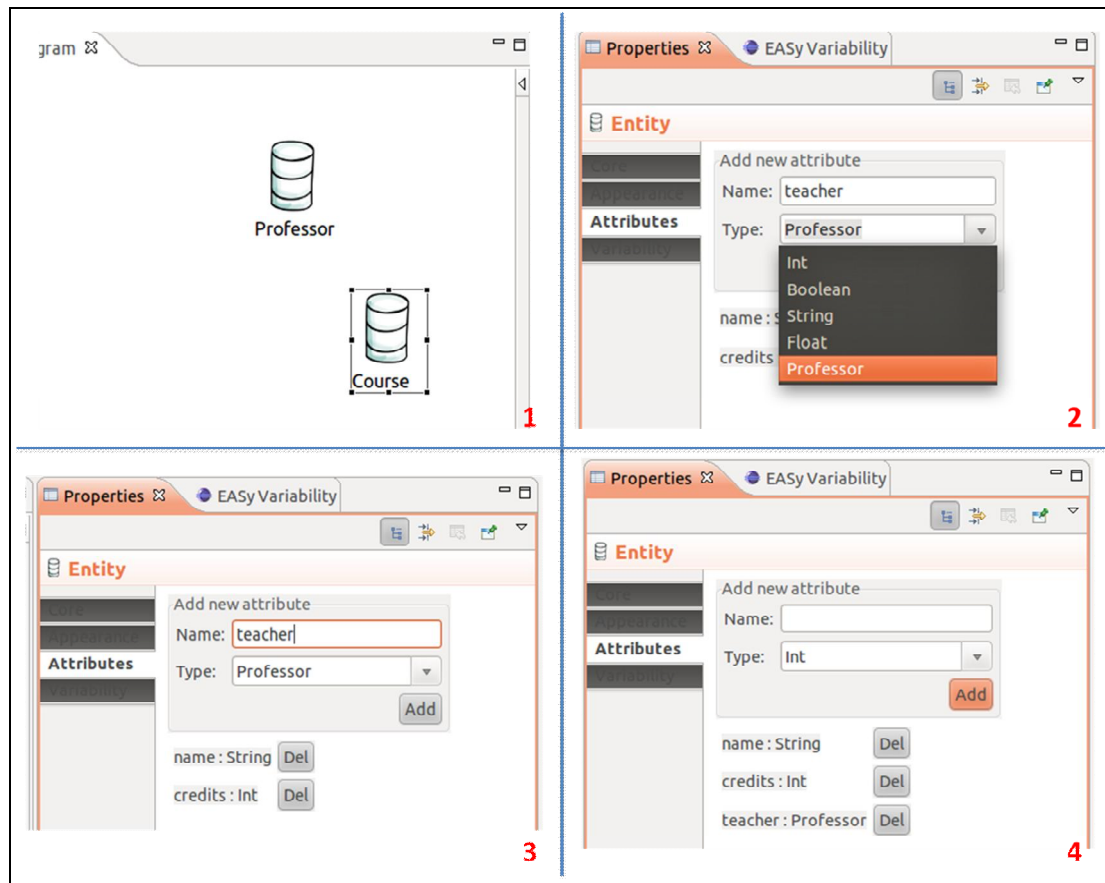


Figure 26: Attributes UI.

The tab is shown only when an instance of Entity or Event class is selected; Figure shows an example: in the model there are two entities, Professor and Course (Figure .1). When Course is selected, in the Property View a tab with label Attribute is shown (Figure .2): at this point the modeller can add variables compiling the form, indicating the name and the type of the attribute. It is worth noting that the difference between PrimitiveAttribute and EntityAttribute is hidden to the modeller: she could manage them in the same way. In Figure .3 the user wants to create an attribute with name "teacher" and type "Professor"; when she presses the Add button IRET creates an instance EntityAttribute associated to the Course entity. As shows in Figure .4, below the new attribute form there is the list of the created attributes: user can delete the attribute through the Del button.

4.1.2 Views

The new version of IRET implements a multi-view mechanism: it allows users to create multiple diagrams related to the same model. This feature is useful in several situations, for example:

- a modeller, in order to manage huge IRENE models, can split it in multiple views;
- multiple users can manage different parts of the same model (i.e. one modeller is responsible of modelling high-level goals capturing the and one more “technical” modeller for the low-level ones);
- manage the needs of several stakeholders maintaining a unique model;

With the introduction of the multi-view mechanism we modified the IRET EMF model, adding three new classes. The schema of the new model is shown Figure .

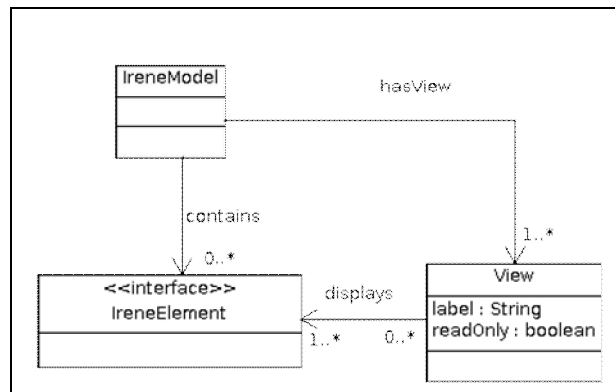


Figure 27: IRET multi-view model.

The new classes are:

- IreneElement: it represents the concepts defined by IRENE (goals, agents, operations, events, attributes, etc.). All the IRENE elements involved in the IRET EMF Model inherits this interface;
- View: it is a container of the instances of IRENE elements shown in a diagram. One view can contains one or more IRENE elements, and one IRENE elements can be related to one or more views. Each view has two attributes: the name and one flag to indicate if it can be modified;
- IreneModel: it represents one IRENE model, it is a container of instances of IRENE elements and views.

While models are stored in files with extension `.iret`, the views are stored in files with extensions `.iret_diagram`. When users create a new IRENE model a default view with the same name is automatically generated (i.e. if user create a model `modell.iret`, then a view `modell.iret_diagram` is automatically generated).

In order to add new views to existing models, users use the dedicated wizard available in the IRET wizard folder (Figure).

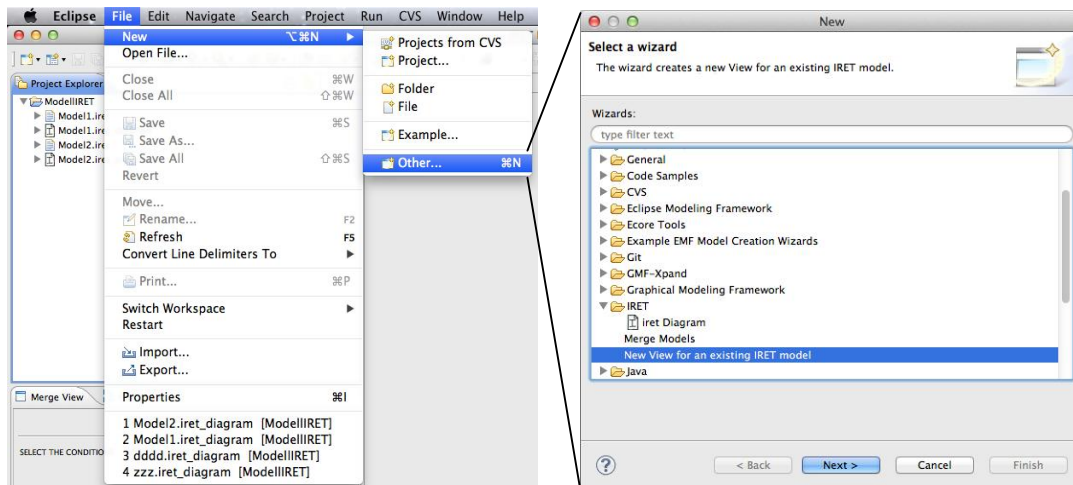


Figure 28: Entering the wizard tools.

Figure 29 shows the main dialog of the wizard: the user should select one existing IRET model in the workspace (through the Select... button) and then she should fill the form indicating the name of the view and if it can be modified.

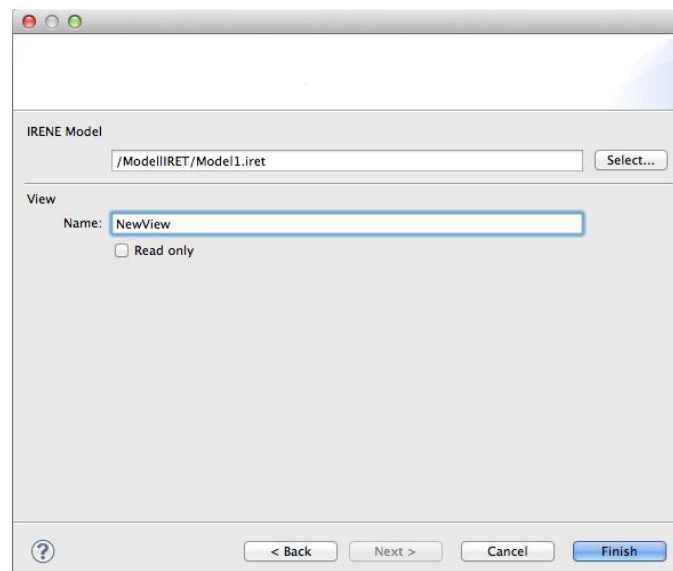


Figure 29: New View wizard: giving a name to the new view.

When the user end the process (pressing the Finish button), a new .iret_diagram files with the name indicated by the user is created (Figure). At this point the user can start to edit the model using both the views.

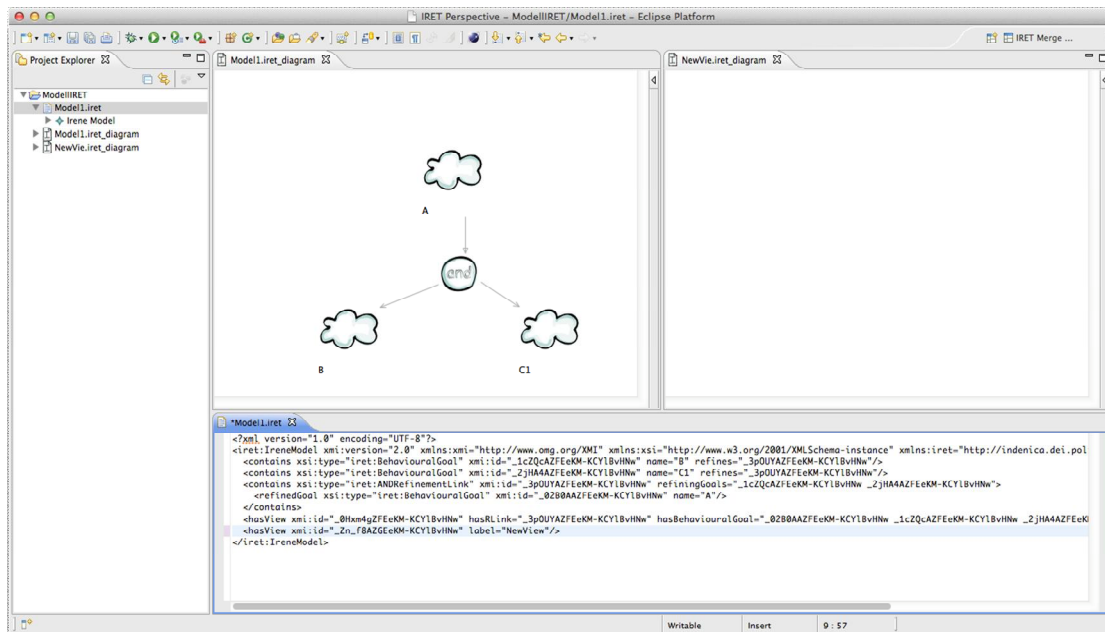


Figure 30: New View wizard: the perspective ready with the new view added to the model.

The new multi-view mechanism requires a method to reuse the existing model components in different views. IRET provides a *drag and drop* feature to achieve this goal.

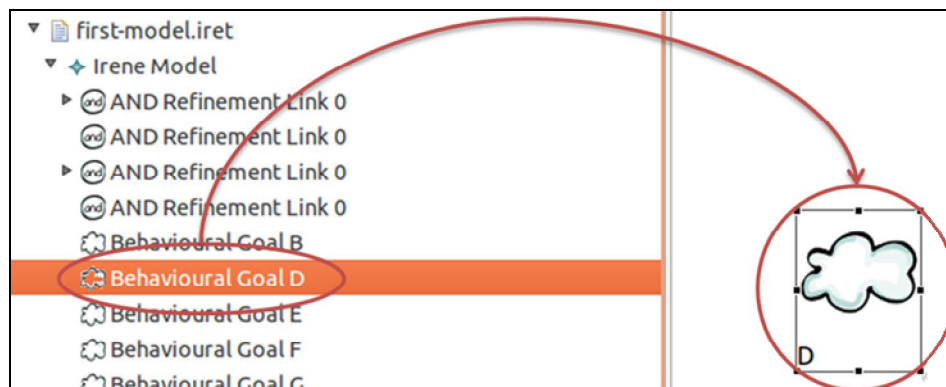


Figure 31: IRET Drag and Drop

As shown in Figure, the navigation panel shows the list of all the available elements of the IRENE model. Users can drag the elements in the drawing area and when they drop it a graphical representation of the element is created in the active view. This view should be one of the available views of the source model (otherwise nothing happens).

4.1.3 Merge

The merge function provides to fuse two input models into a single one as the result of a semantic analysis process between the nodes. A specific Eclipse perspective² was developed for this purpose. The Figure shows how to switch to this perspective.

² In the Eclipse Platform a *Perspective* determines the visible actions and views within a window providing mechanisms for interaction with resources, multi-tasking and information filtering

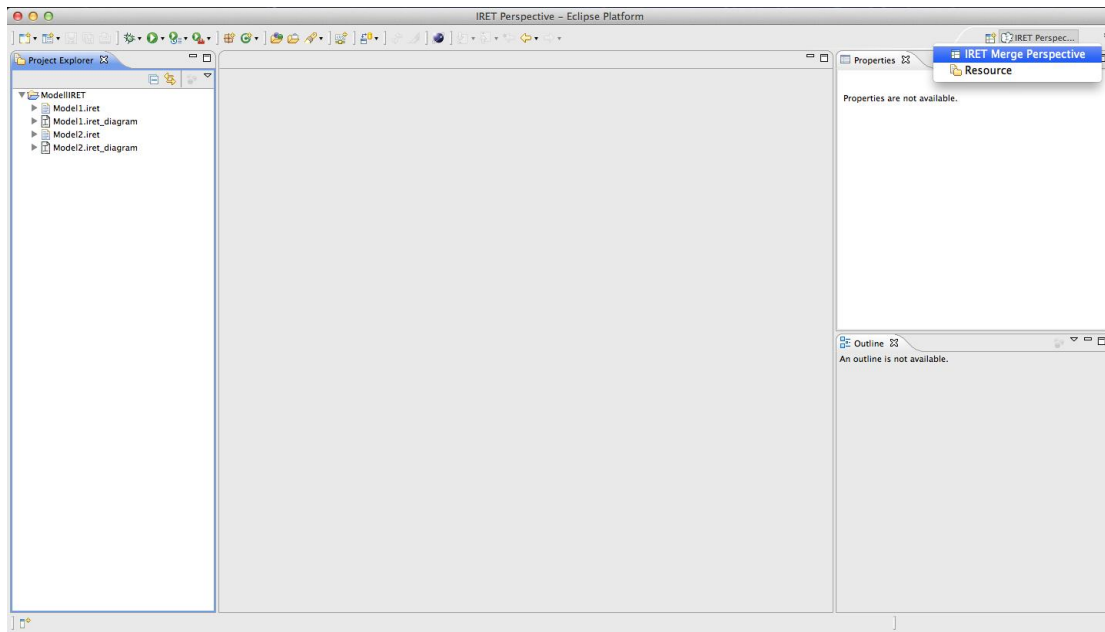


Figure 32: Switching to the Merge Perspective.

When the modeler switches to the IRET Merge Perspective a new panel appears on the left bottom corner as shown in Figure 33.

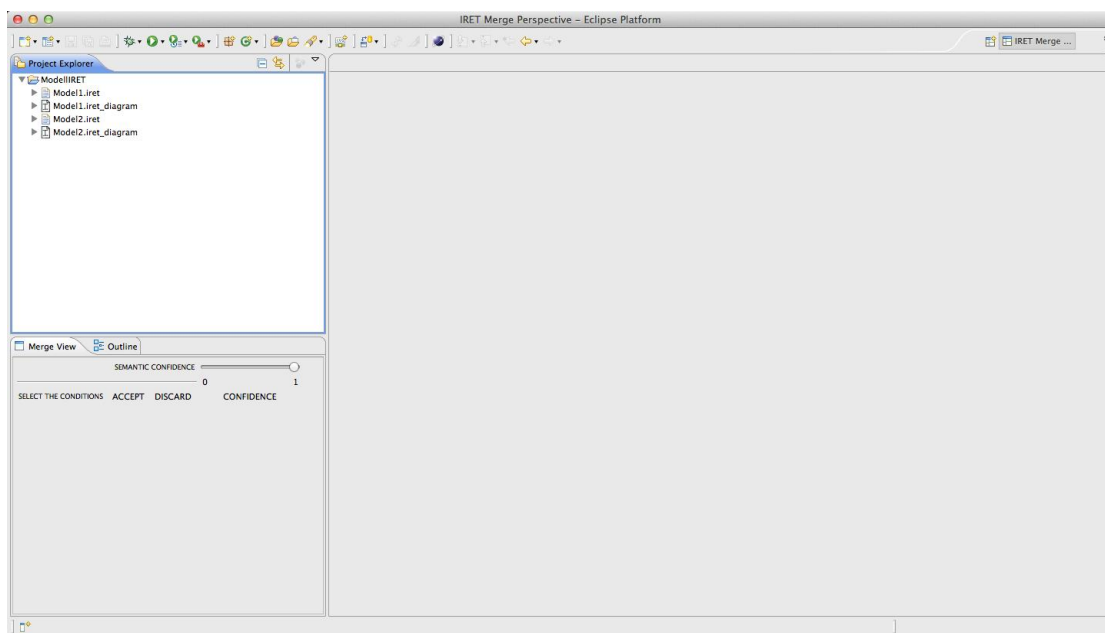


Figure 33: The Merge Perspective

In order to start the merge operation the modeler uses the *New* function in the Eclipse menu and selects *Merge Models* under the *IRET* section, as shown in Figure .

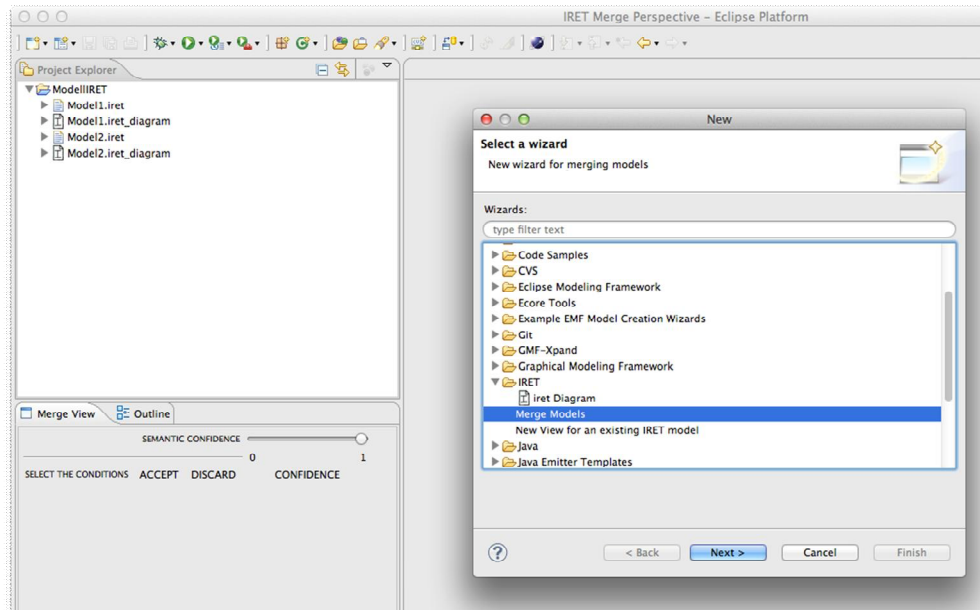


Figure 34: How to launch the Merge models wizard.

The merge operation is organized in steps. The first step is to choose the two input models to be merged and the name of the output model.

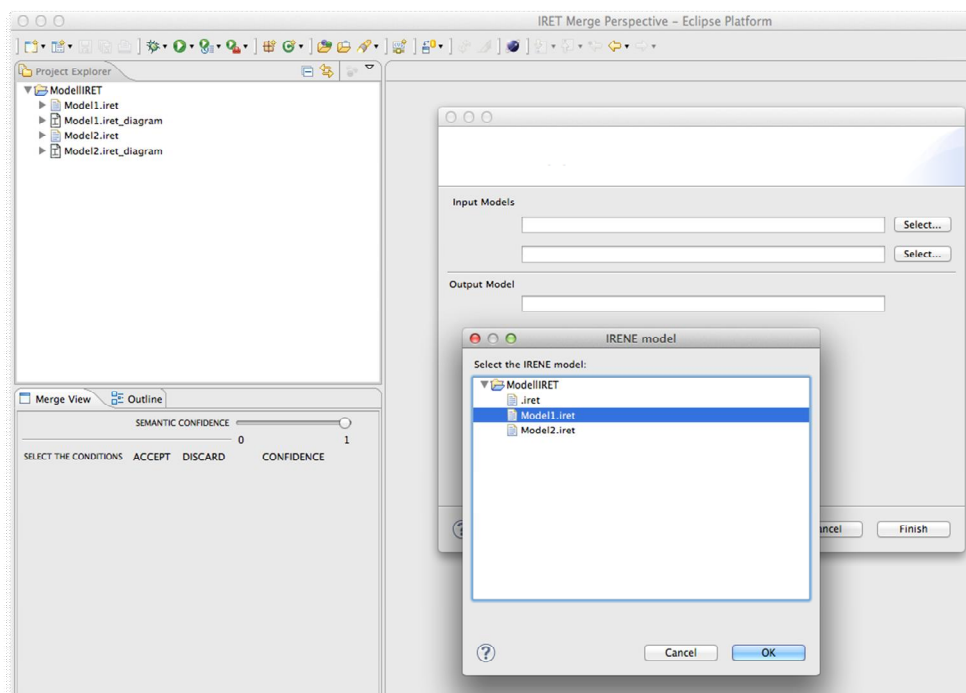


Figure 35: Merge models wizard: selecting the Input Models.

The first model will be the one with higher priority: i.e. in case of two nodes named "A" and "B" with the same meaning (high semantic proximity), the output model will contain a merged node named "A".

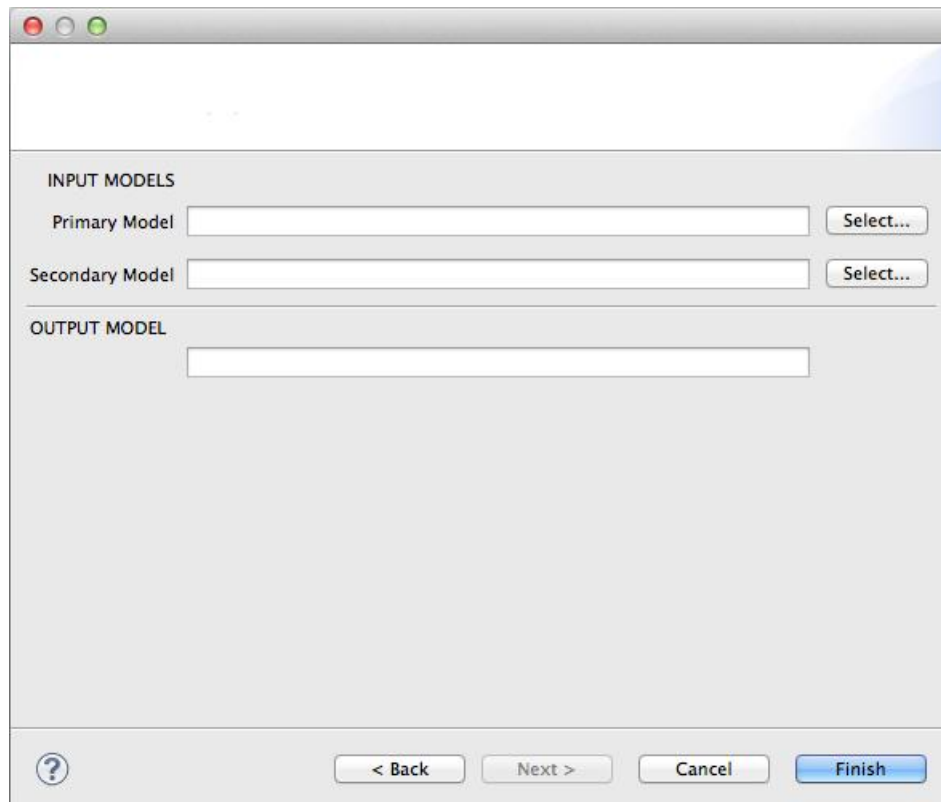


Figure 36: Merge models wizard: giving a name to the Output Model.

By pressing the Finish button the analysis process starts. The nodes of the two models are compared and the similarity between nodes is evaluated. The similarity is based both on the semantic proximity of the names of each node and on the priority of each node as assigned by the modeler in the original model.

The name of any entity could be any sentence in natural language. The aim of the comparison process is to obtain the semantic proximity expressed as a number in a range from 0 to 1. In this version of the prototype the semantic proximity evaluation is based on the vocabulary defined by [WordNet]. The implementation semantic comparison algorithm is based on a third part Java library [JavaSimLib]. This library returns a numeric value in a range 0..1 that represents the degree of similarity between two words in English. As mentioned the similarity between the names of the nodes is then normalized taking into account the priority of a node (and of the underlying requirement or goal) in a model. This improves the quality of the result of the merge operation. In fact comparing complex graphs can lead to conflicts that have to be solved by the modeler and the information about the priority of the nodes in the original model supports the right choice.

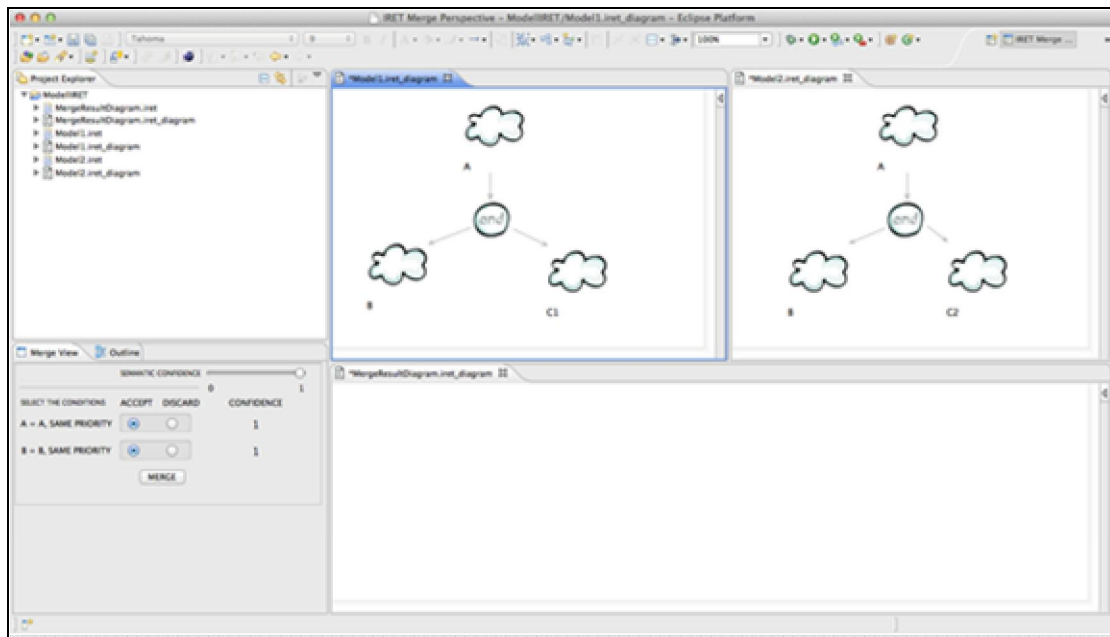


Figure 37: Merge models wizard: the Perspective ready for merge output

The result appears on the Merge View panel. The user can decide to accept or discard each of the suggestions about a possible similarity proposed by the tool. If the user does not confirm a proposed similarity on a couple of nodes he/she force the algorithm to consider them as two different nodes in the result graph.

As we mentioned the semantic proximity is expressed in a 0..1 range, thus IRET introduces the concept of Semantic Confidence and offers to the modeler a scale regulator the user can play with. Setting the regulator to 1 means that the modeler can confirm only merges between nodes that have strict semantic proximity. Thus, only couples of nodes that obtain the highest index from the comparison algorithm are considered nodes. Setting the regulator to 0 imply that the algorithm does not perform any comparison and considers any node belonging to the first input model as different from any other in the second model.

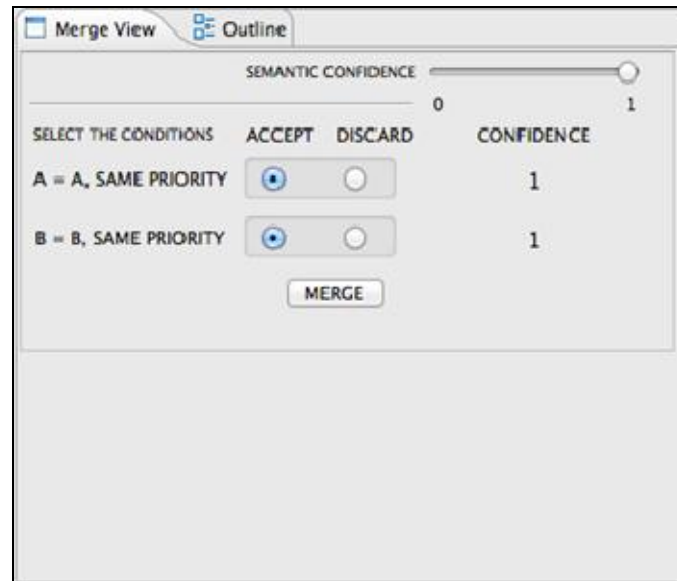


Figure 38: Merge models wizard: the Merge Panel with the controls for tuning the merging algorithm.

The tuning operation affects the merging process. The MERGE button should be pressed when the confidence and the conditions are set.

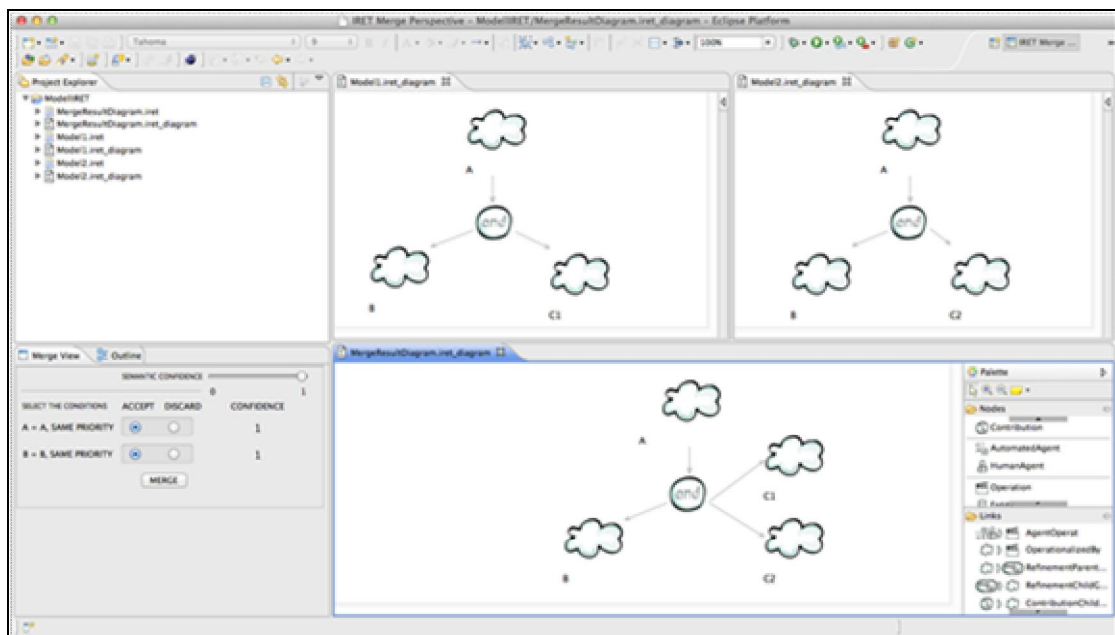


Figure 39: Merge Models wizard: the Merge Perspective with the merge model result.

The merge algorithm is quite simple in its logic. When two nodes are considered equals (and this condition is accepted on the merge panel) the one coming from the first model will be included in the merge output model. Each sub-tree belonging to the two merged nodes is included and linked to this new resulting node. The link is realized through an "OR" operator into the output model. The role of this operator is to preserve the integrity of all the paths originally structured on the input models. The final output is the iteration on all the equal nodes on the merged model, so that any duplicated node is reduced to a single one.

4.1.4 Other improvements

Additionally to the main features described in the previous sections, IRET has a set of minor new features. These improvements were realized to address suggestions and requests of the IRET users and other INDENICA WPs; we will briefly present the most relevant ones in the following:

- New wizard for model creation: a simplified wizard to create an IRENE model has been introduced. Instead of selecting in two steps the name of the model and the name of the view, now the user may insert only the name of the model. Now when user creates a model a view with the same file name is created automatically;
- New graphical elements: one of the most common feedback that we received after the first release of IRET (as described in D1.2.1) was the difficulty in finding the elements in the palette. In order to address this comment we developed set of new icons (Figure 40) to replace the default ones. The new graphical elements help in understanding faster the meaning of nodes and links that can be used in the diagram, improving the usability of the system.

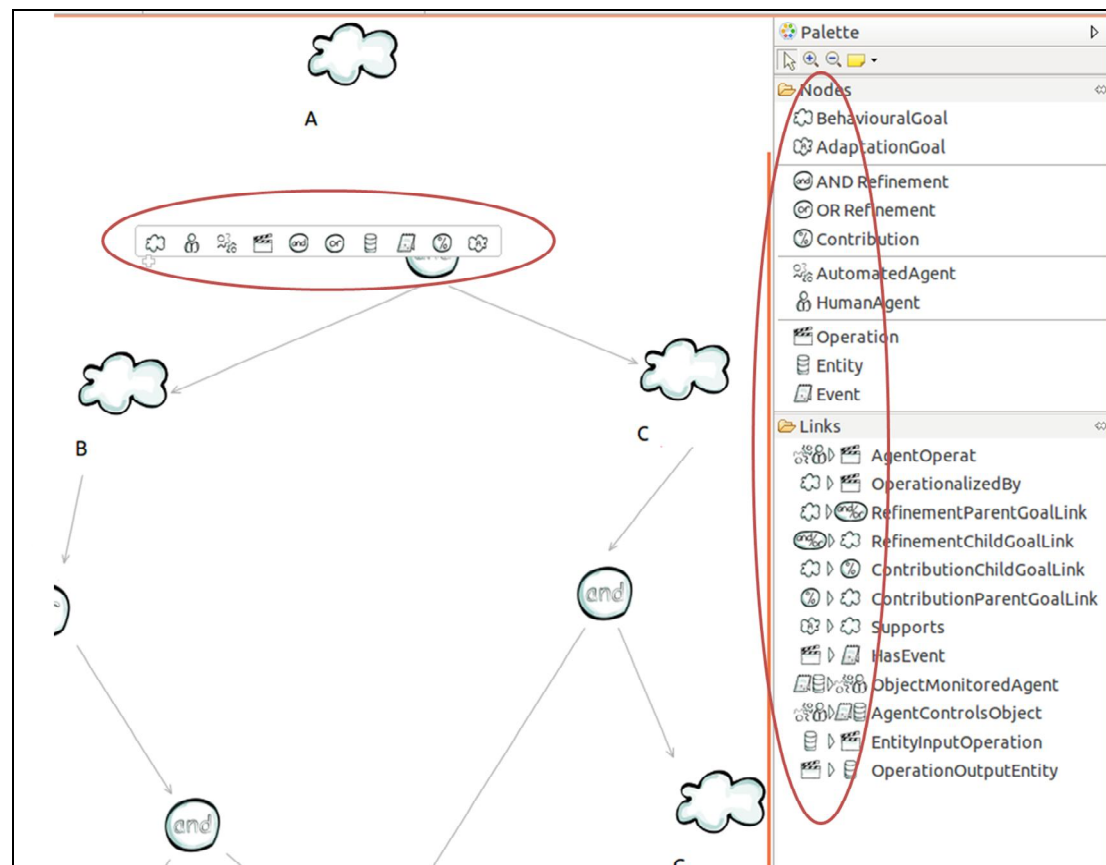


Figure 40: New graphical elements for IRENE elements

- Copy and paste: The need of reuse existing elements such as goals or actors in different model is quite common. For this reason, another request we often received was the feature of copy and paste elements between models. We implemented it, enabling IRET to copy and paste elements between diagrams. It is worth noting that the copied element doesn't maintain any relation with the

original element: this operation creates a “clone” of the original element in the target model.

4.1.5 Integration with other tools

IRET is integrated with other two INDENICA tools: the Priorization Tool Support and EASy. The first application aims to determine the priority of the requirements and it consumes as input IRENE models. We will go in depth on this topic in Section 4.2.

On the other hand EASy is a modelling tool developed by WP2 to allow INDENICA users to define variability models. In general goal modelling and variability definitions are two parallel activities: it means that we should provide some features to help users to do both the task at the same time. It is the main goal that the IRET-EASy Bridge aims to achieve. The tool has been developed by WP1 in cooperation with WP2.

By a technical point of view, both IRET and EASy are Eclipse plug-ins; this fact helps in the integration process through the exploitation of the features offered by the Eclipse framework.

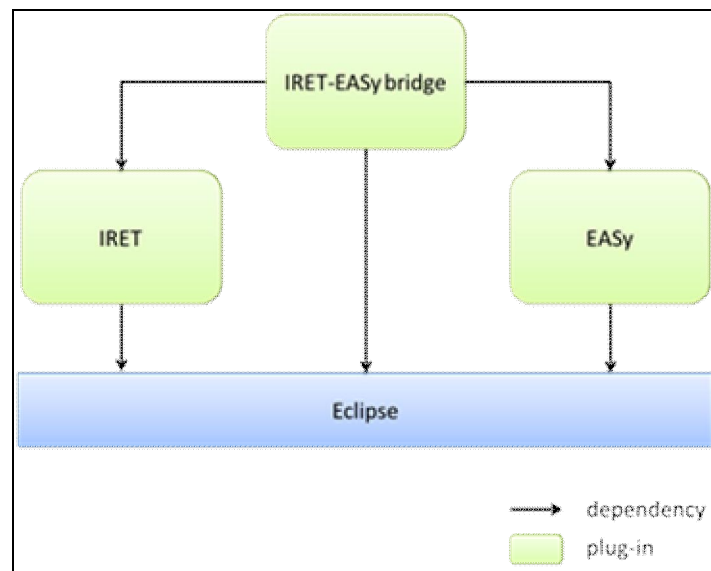


Figure 41: IRET-EASy Bridge architecture

Figure shows the high level architecture of the integration: IRET and EASy are two Eclipse plug-in that can be installed independently the one from the other (i.e. it is possible to install only IRET or only EASy); IRET-EASy Bridge is another Eclipse plug-in that depends from both IRET and EASy. It means that it requires both the tools in order to work. Detailed instructions on how to install the bridge plug-in are available in the Appendix.

By a graphical point of view the tool introduces a new View, named EASy Variability (Figure).

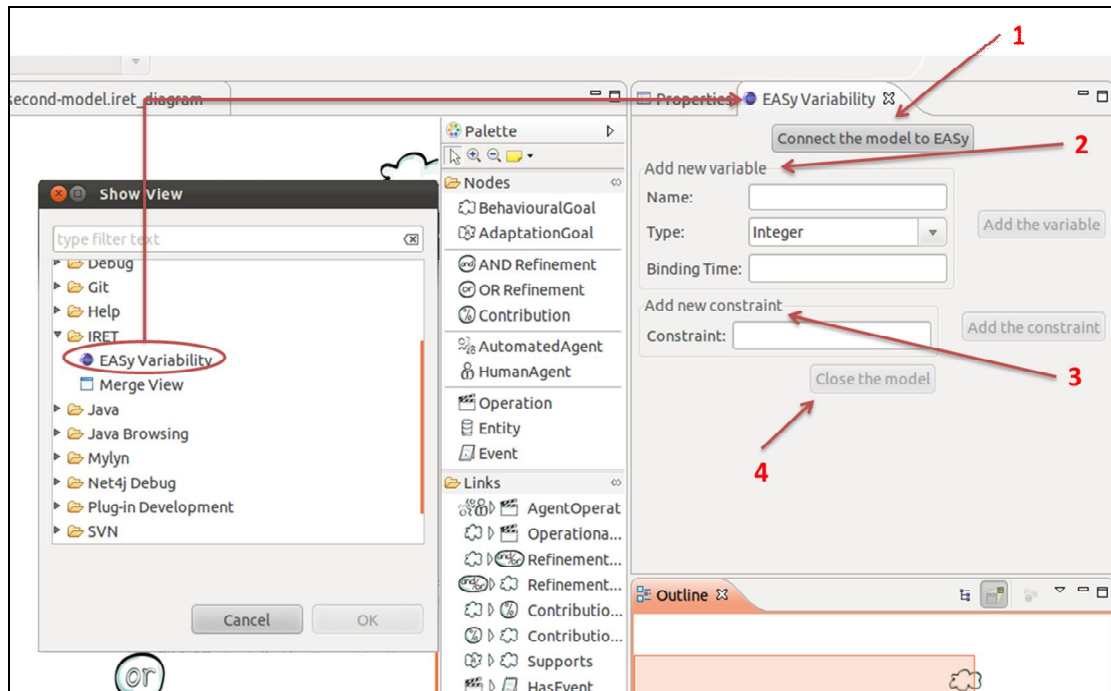


Figure 42: IRET-EASy Bridge view.

The view is opened in the IRET perspective and allows users to control the EASy tool from IRET. The interface offers four operations:

1. Connection to EASy: the button opens a connection with the EASy-producer. It is the first step required to add the variable and the constraints;
2. Variable definitions³: through this form is possible to add new variables to the variability model. The three required fields to define a variable are:
 - The name of the variable;
 - The type of the variable (Integer, String, Boolean, etc.);
 - The binding time (compile time, run time, etc.);
3. Constraint definitions³: the form is used to define and submit a new constraints to the variability model;
4. Close the model: the last button closes the connection and starts the generation of the EASy variability model. After that this button is pressed the EASy perspective is launched, allowing the user to use the user interface provided by the WP2 tool.

Variables and constraints are also added to the IRET file: the IRENE model has been extended in order to enable the storage of this information (Figure).

³ Additional information about EASy and its data model are available in [TBC].

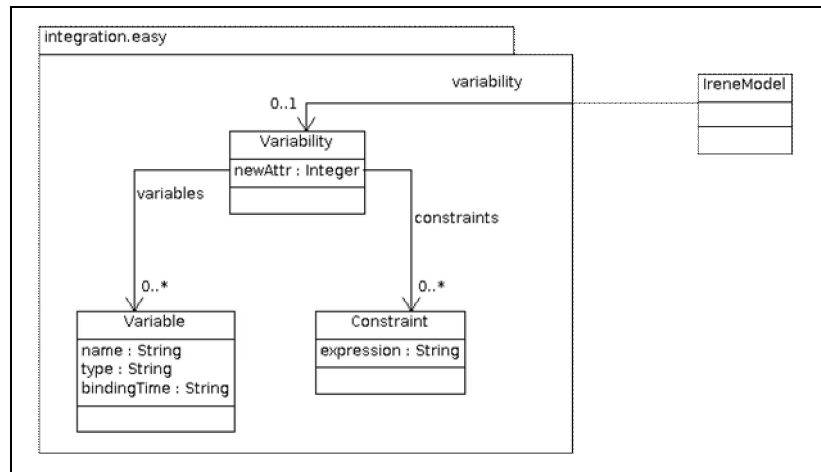


Figure 43: EASy extension to the IRENE model.

The extension has been made in order to introduce a “save and restore” mechanism: users can compose their models (the IRENE and the EASy ones) in several sessions, closing the EASy model (operation 4) only when they finish the goal definition. The extension is also a required basis to research on a more complex and automated integration between the two methodologies (i.e. automatically infer parts of the EASy model when user define elements of the IRENE model).

4.2 Prioritization Tool Support

The tools for prioritization support were written in Java using Eclipse Indigo development environment and a number of Java libraries:

- JDOM for parsing XML files [jdom]
- Java Excel API for reading and writing MS Excel files [jexcelapi]
- JFreeChart for generating graphics [jfreechart]

4.2.1 Goal Extractor

In order to extract goals from a goal model for prioritization start the program Extract_Goals. In the first window you can select a number of IRET-files:

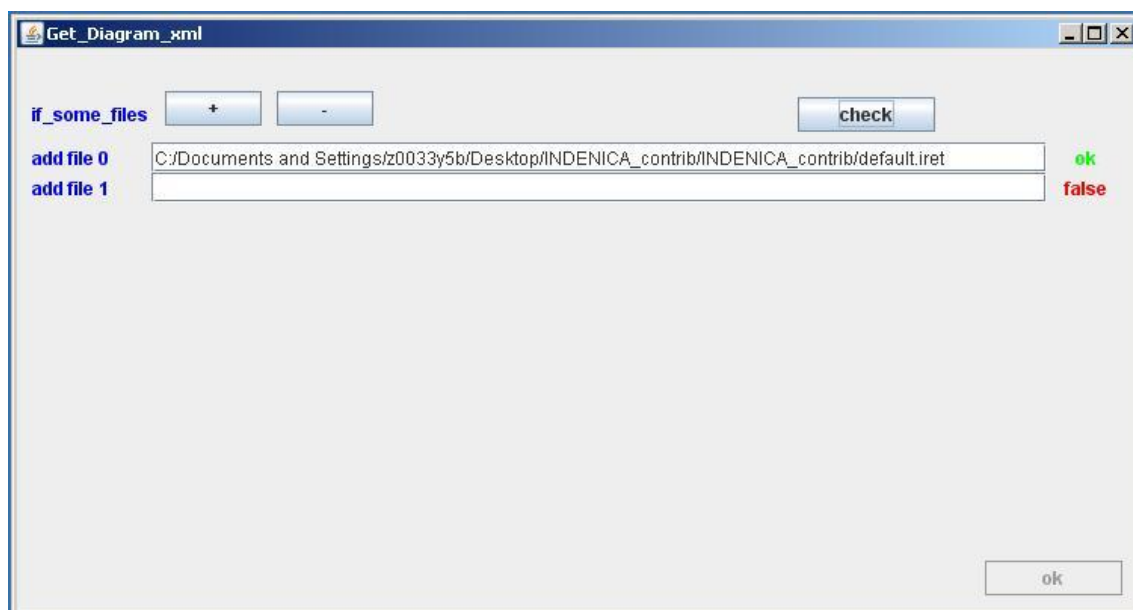


Figure 44: Goal Extractor: Selection of files.

With the button “Check” it is ensured that the path- and filename points an IRENE goal model XML file (file type *.IRET). With buttons “+” and “-” you can add or remove files from the list.

If all files are checked and marked “ok” press “ok” to continue. The selected XML files are parsed now and a list of the extracted goals is shown:

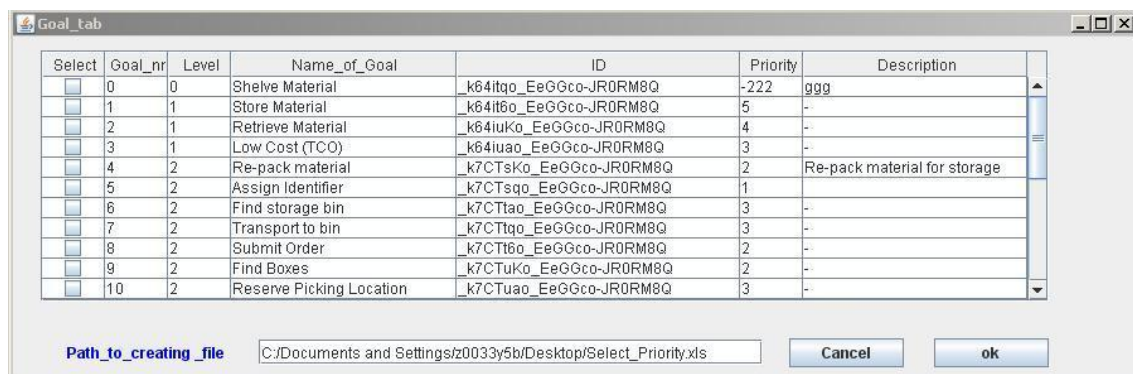


Figure 45: Goal Extractor: list of goals for selection.

The list of goals has seven columns.

- Select: here you can tick the box for selecting a goal for further processing
- Goal_Nr: sequence number of goals found in the IRET files
- Level: position of the goal in the goal hierarchy; 0 is the root goal, the highest number indicate the leaves
- Name: name of the goal as entered in IRET
- ID: internal goal ID in IRET (needed for identification when writing back priorities)
- Priority: priority of goal if entered in IRET; note that IRET does only allows numbers, but does not apply any other rules
- Description: as entered in IRET

The strategy for selecting the goals to be prioritized was discussed in chapter 3.5.2.

After pressing the button "OK" an Excel file is generated as entered in the field Path_to_creating_file.

	A	B	C	D	E	F	G
1	Goal_nr	Level	Name_of_Goal	ID	Priority	Description	
2	0	1	Low Cost (TCO)	_ohcCh-0CEeGNS7u-_dlH0A	1	-	
3	1	2	Re-pack material	_ohcCio0CEeGNS7u-_dlH0A	1	Re-pack material for storage	
4	2	2	Assign Identifier	_ohcCiu0CEeGNS7u-_dlH0A	3		
5	3	2	Find storage bin	_ohlMc00CEeGNS7u-_dlH0A	2	-	
6	4	2	Transport to bin	_ohlMce0CEeGNS7u-_dlH0A	2	-	
7	5	2	Submit Order	_ohlMcu0CEeGNS7u-_dlH0A	1	-	
8	6	2	Find Boxes	_ohlMc-0CEeGNS7u-_dlH0A	1	-	
9	7	2	Reserve Picking Location	_ohlMd00CEeGNS7u-_dlH0A	2	-	
10	8	2	Transport Boxes to P.L.	_ohlMde0CEeGNS7u-_dlH0A	3	-	
11	9	2	Pick Items from Box	_ohlMdu0CEeGNS7u-_dlH0A	3	-	
12	10	2	Prepare Shipping Container	_ohlMd-0CEeGNS7u-_dlH0A	2	-	
13	11	2	Ship Container	_ohlMe00CEeGNS7u-_dlH0A	1	-	
14	12	2	Low nuber of staff	_ohlMee0CEeGNS7u-_dlH0A	1	-	
15	13	2	High Storage Density	_ohlMeu0CEeGNS7u-_dlH0A	1	-	
16	14	2	Low Energy Consumption	_ohlMe-0CEeGNS7u-_dlH0A	2	-	
17	15	1	High Throughput	_ohlMfo0CEeGNS7u-_dlH0A	1	-	
18							
19							

Figure 46: Excel file with selected goals.

This file can then be distributed to the involved user groups. For better readability it is helpful to hide columns B and E as they are not relevant for the users but for internal processing only.

4.2.2 Prioritization Analyzer

After collecting the Excel files form all users that have done their voting store all files in a dedicated directory and start the program "Prioritize_Goals".

The first window is for selecting the files for comparison.

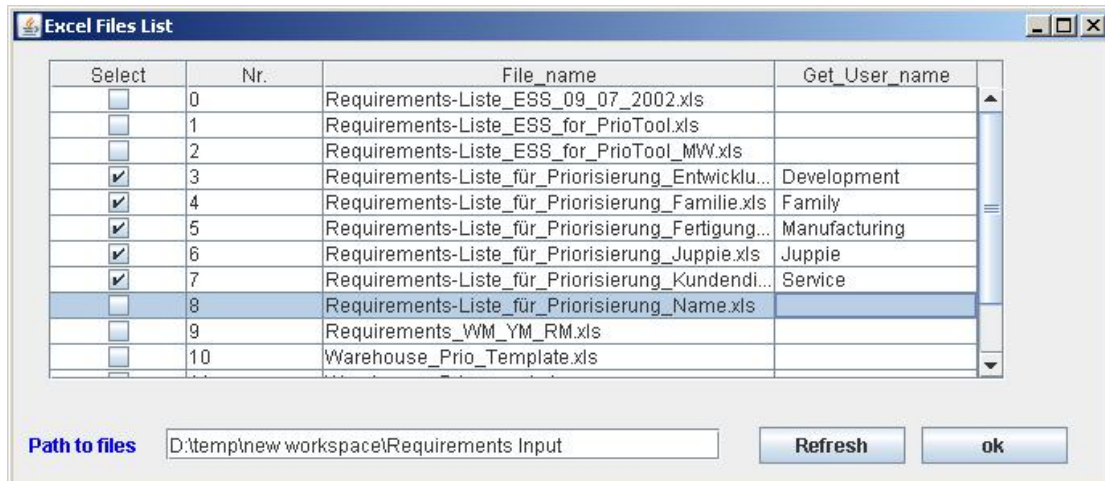


Figure 47: Selecting prioritization files .

Enter the correct path of the directory and press "Refresh". A list of all *.xls files is shown. Mark the files for comparison in column one.

As an option you can enter a user / user group name in column four for each file. If left free, in following windows the default names Usr_0, Usr_1, Usr_2 ... will be used.

After pressing OK the files are read and all the prioritization values are shown in a matrix.

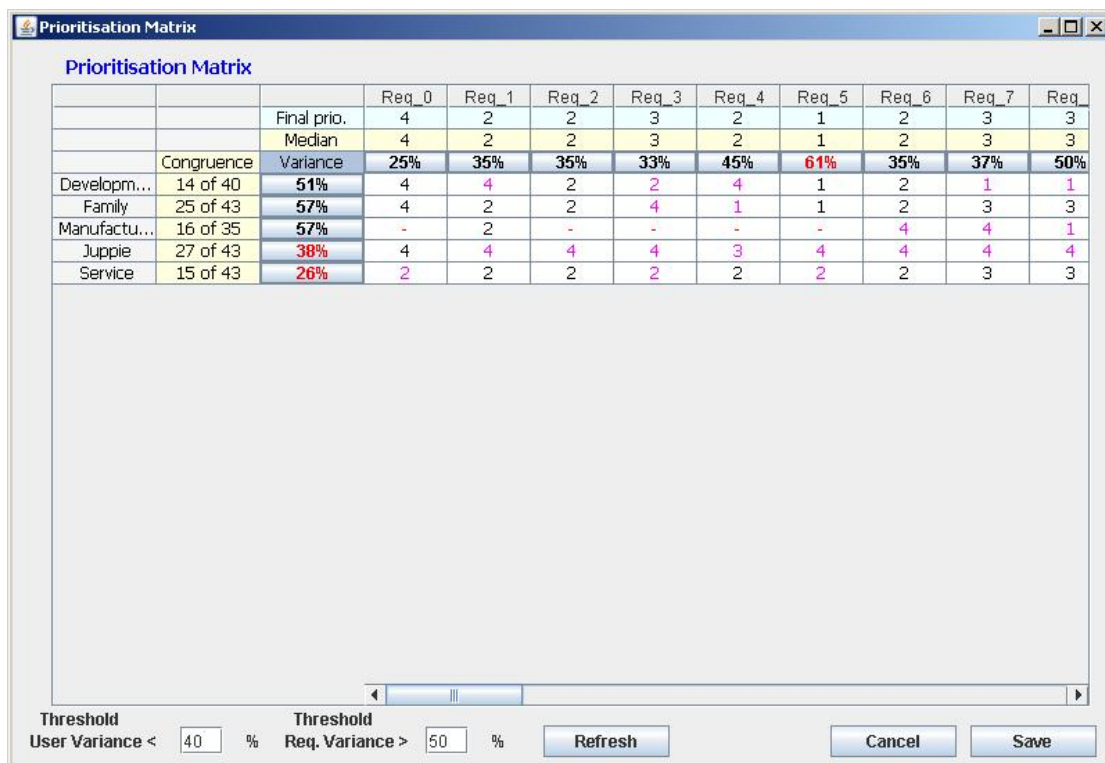


Figure 48: Prioritization Matrix.

The matrix shows all user priority voting. In case a user did not vote or entered 0, a dash is in the respective cell. If the use voting is not equal to the median, the vote is written in red.

Row 1 shows the requirement (or goal) number. When hovering with the mouse over it, all information for a requirement is shown in a pop up box.

Row 2 shows the final priority, which is in the initial state of the matrix equal to the median in row 3.

Row 3 shows the median value of votes on a requirement.

Row 4 shows the variance of user voting on the respective requirement. If this number is red, it is above threshold in the field "User Variance threshold" on the bottom of the window. The default value is 40% and it can be changed by entering a number and pressing the button "Refresh".

With double click on the variance field, the window "Priorities per Requirement" opens for further analysis (see below).

Column 1 shows the user name entered in the field selection box

Column 2 shows the congruence of the user's voting, which is the number of voting that are equal to the median.

Column 3 shows the variance of the user's votes. If this number is red, it is above threshold entered in the field "Requirements variance threshold" on the bottom of the window. The default value is 50% and it can be changed by entering a number and pressing the button "Refresh".

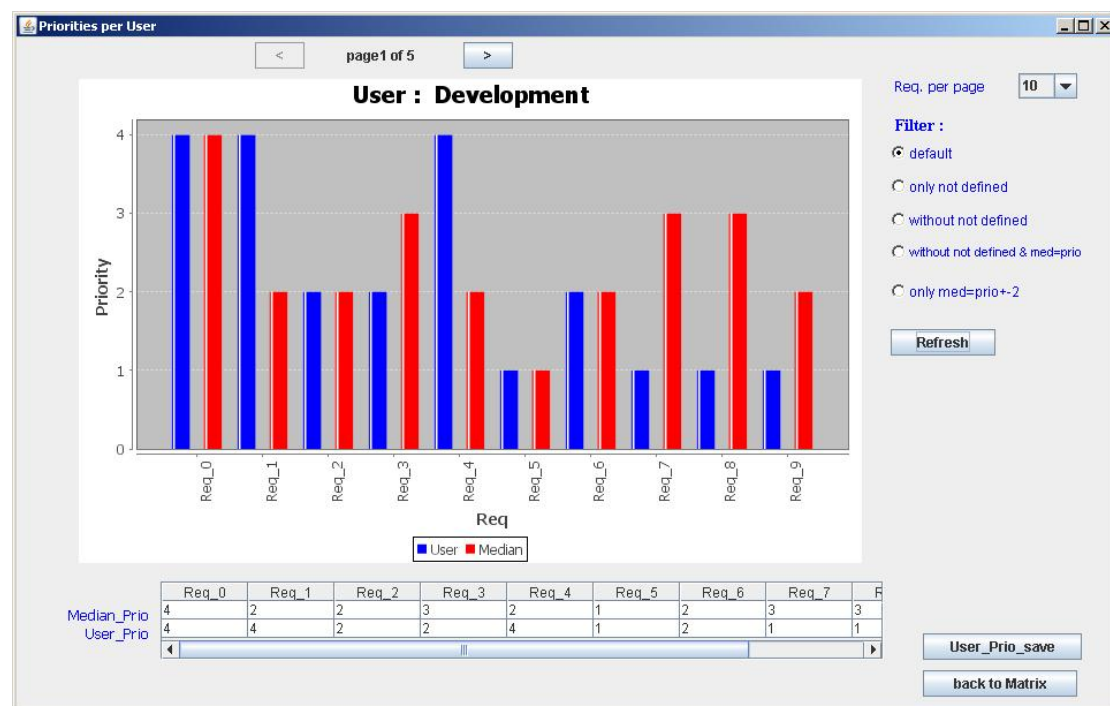


Figure 49: Priorities per User.

The window "Priorities per User" allows a deeper insight in the priority voting of a single user. On the right side there are options to limit the number of requirements displayed and on the top line there are scroll arrows to see next or previous set of requirements.

Further there are a number of filters for selections:

- Normal: shows all requirements
- Only not defined shows all requirements on which the user did not vote
- Without not defined shows only requirements on which the user did vote
- Without not defined & med=prio shows all voting of the user which are not equal to the median
- Only med=prio+-2 shows all votes where the user voting differs with 2 or more from the median.

With these options it is easy to identify user priority voting that should be further analyzed and discussed.

If the user agrees to change his priority vote, it can be changed by clicking on the vote in the table under the graphic and select the new priority vote.

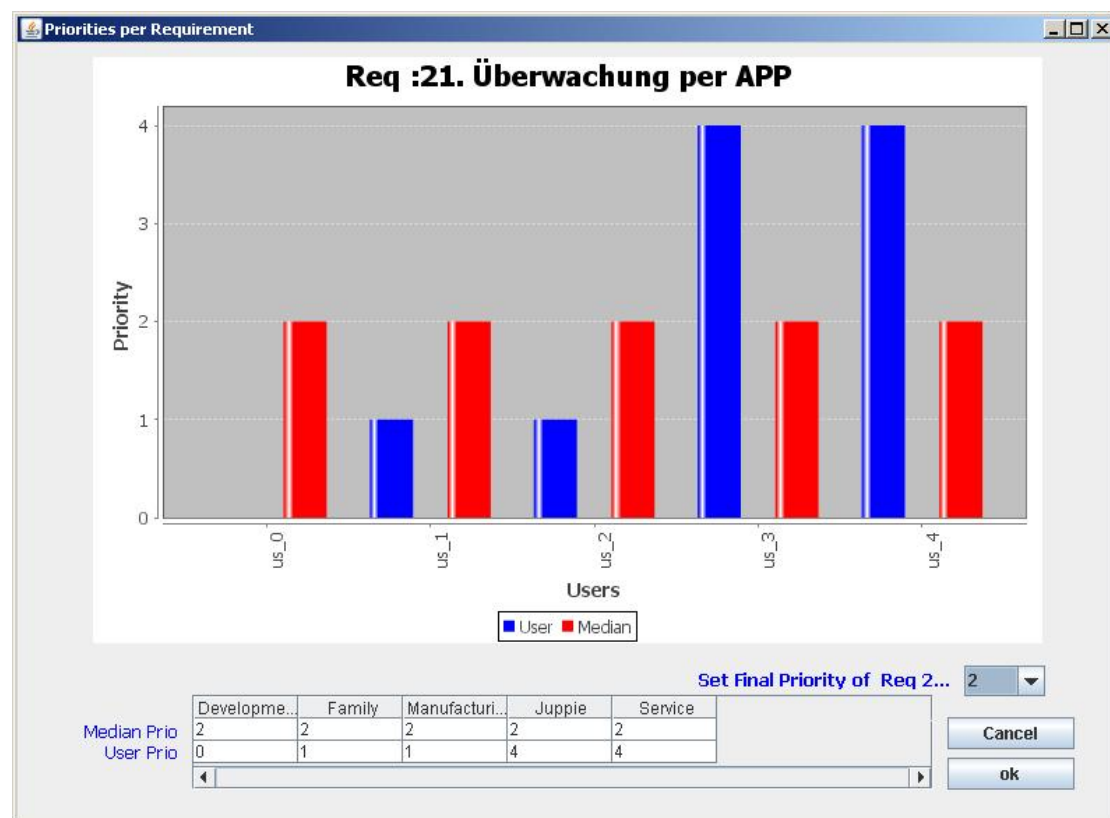


Figure 50: Priorities per Requirement.

The window "Priorities per Requirement" shows all the user priority votes on a single requirement. Its name is written on the top of the window.

Below the graphic a table shows the numerical values. A drop down list allows setting the final priority of the requirement that was agreed upon. The default value is equal to the median.

When clicking "OK" the final value is written to the matrix and the Prioritization Matrix (Figure) is shown with the updated values.

When clicking “Save” the final priority, an IRET goal model file can be specified, where the goals were extracted and the priorities are written into this model using the goal IDs. If the goals cannot be identified a warning is issued.

5 Conclusions

This document adds to and complete the work initially presented in D12.1. After the first two years of the project, this is also the end of the theoretical work on the INDENICA solution for requirements elicitation and specification. In these years, the work proceeded according to some parallel threads: (a) the work on defining the elicitation process and methods, (b) the work on ROI calculation, (c) the work on IRENE, the goal-oriented specification notation, (d) the work on composing conflicting priorities, and (e) the development of the different supporting tools. As further dimension, we can also mention the work done by our users and demonstrators, who started working on the first ideas and versions of the proposed solution, and with their comments, suggestions, and bug reports helped us improve final set of methods and tools.

Now that the INDENICA solution for requirements elicitation comprises a coherent set of methods and tools, and that it addresses the problem of eliciting the different requirements for service platforms from different viewpoints, it is time to move to the next phase. Within the project, users and demonstrators are required to keep working on these methods and tools, and provide further comments and a more thorough evaluation. Outside the project, it is now time to present these ideas, along with their supporting tools, to the international community (e.g., through focus groups, industry-oriented seminars, and participation to the main academic conferences in the field).

6 References

- [Anton, Potts 2009] Annie I. Anton, Colin Potts: The Use of Goals to Surface Requirements for Evolving Systems (Scientific Literature Digital Library and Search Engine (United States) 2009.
- [DeBaud 1999] J-M DeBaud, K. Schmid: A Systematic Approach to Derive the Scope of Software Product Lines. [ICSE '99](#) Proceedings of the 21st international conference on Software engineering 1999
- [Evans 2002] Evans, Meryl K.: Understanding UCD (Interview with Peter Merholz and Nathan Shedroff) http://www.digital-web.com/articles/peter_merholz_and_nathan_shedroff (accessed May 25, 2011.
- [Sun 2007] Hongquing Sun: Developing User-centric Software Requirements Specifications. Master Thesis at the Mc Master University, Hamilton, Ontario, 2007
- [Kang et al. 1990] K.Kang, S. Cohen, J. Hess, W. Novak and S. Petersen: Feature-Oriented domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, SEI, Carnegie Mellon Univ., Nov. 1990.
- [Kreuter2008] Kreuter et al.: Applying a Cost Model for Product Lines: Experience Report. MESPUL 2008.
- [Krueger2007] Krueger: The 3-Tiered Methodology: Pragmatic Insights from New Generation Software Product Lines: Proceedings of the 11th International Software Product Line Conference, IEEE Computer Society, 2007
- [Mikyeong2005] M. Mikyeong, Y. Keunhyuk, S.C. Heung: An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in a Product Line: IEEE Transactions on software engineering, Vol. 31, No. 7 ,July 2005
- [Schleicher2004] W. Schleicher: Domain Analysis und Scoping: Seminar Produktlinien, Uni Stuttgart WS 2003/2004
- [Sutcliffe1998] Alistair G. Sutcliffe: Scenario-based Requirements Analysis, Requirements Engineering Journal, Vol. 3, pp. 48-65, 1998
- [Voelter2006] Stahl, Voelter: Model-Driven Software Development Technology, Engineering, Management: Wiley, 2006
- [Pohl2005] K. Pohl, G. Böckle, and F. v. d. Linden, Software Product Line Engineering Foundations, Principles, and Techniques. Berlin: Springer, 2005.
- [Clements2005] Clements, McGregor, Cohen: The Structured Intuitive Model for Product Line Economics (SIMPLE):, Technical Report, CMU/SEI-2005-TR-003, ESC-TR-2005-003, 2005
- [Böckle2004] Böckle et al.: Software Produktlinien: Methoden, Einführung, Praxis, dpunkt-Verlag 2004.

[Bosch2002] Bosch: Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization, Proceedings of the Second Conference Software Product Line Conference (SPLC2), pp. 257-271, August 2002.

[Bosch2000] Bosch: Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach: Addison Wesley, 2000

[Bayer1999] Bayer, Flege, Knauber, Laqua, Muthig, Schmid, Widen, DeBaud: PuLSE: A Methodology to Develop Software Product Lines: Proceedings of the 1999 symposium on Software reusability, pp. 122 – 131, 1999

[STARS 1996]: Informal Technical Report for "Software Technology for Adaptable, reliable Systems (STARS)". *Organization domain Modelling (ODM) Guidebook, Version 2.0, June 1996.*

[SRI 1995]: Department of Defense - Software Reuse Initiative, Version 3.1. Domain Scoping Framework, Volume 2: Technical Description, 1995

[Lauenroth et al. 08] K. Lauenroth, K. Pohl. "Dynamic Consistency Checking of Domain Requirements in Product Line Engineering," Proceedings of the 16th International Requirements Engineering Conference (RE '08)., pp.193-202, 2008.

[Pnuli 77] A. Pnuli. The Temporal Logic of Programs. In 18th Symposium on Foundations of Computer Science (FOCS), pages 46–57, 1977.

[ReleasePlanner]

[van Lamsweerde 2009] Axel van Lamsweerde. Requirements Engineering: From System Goals to UML Models to Software Specifications. John Wiley, 2009.

[Hoffmann 2011] Hajo Hoffmann, Requirements Engineering & Scrum, Systemanalyse im agilen Umfeld; in: Marc Sihling, Andreas Rausch, Christian Lange, Marco Kuhrmann (Hrsg.) Software & Systems Engineering Essentials, Proceedings 2011

[Mikeyong, Keunhyuk et al. 2005] Mikeyong Moon, Keunhyuk Yeom and Heung Seok Chae. An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in a Product Line in IEEE Transactions on Software Engineering, Vol. 31, No. 7, July 2005.

[Potts et al. 1994] C. Potts, K. Takahashi, A. Antón: Inquiry Based Requirements Analysis, IEEE Software, March 1994.

[PuLSe] see www.software-kompetenz.de , the use search function for "PuLSe", accessed 23. August 2012

[Lennon et al. 1988] M. Lennon, D. Pierce, B. Tarry, P. Willett, An evaluation of some conflation algorithms for information retrieval, Journal of Information Science 8 (3) (1988) 99–105.

[Seco et al. 2004] N. Seco, T. Veale, J. Hayes, An intrinsic information content metric for semantic similarity in Wordnet, in: Proc. European Conf. on Artificial Intelligence (ECAI'04), Valencia, Spain, August 22-27, IOS Press, 2004, pp. 1089–1090.

[JavaSimLib] JavaSimLib – <http://kenai.com/projects/javasimlib>

[jdom] JDOM for parsing XML files, available via: <http://www.jdom.org/>

[jexcelapi] Java Excel API for reading and writing MS Excel files, available via:
<http://jexcelapi.sourceforge.net/>

[jfreechart] JFreeChart, available via <http://www.jfree.org/jfreechart/>

[EMF] Eclipse Modeling Framework Project (EMF) -
<http://www.eclipse.org/modeling/emf/>

[WordNet] <http://wordnet.princeton.edu/>

Appendix

We have successfully tested IRET on Eclipse 3.6 (Helios) and 3.7 (Indigo) on the following operating systems:

- Ubuntu 11.04
- Windows 7

6.1 Installation

- Download the IRET zip file from this site:
<https://repository.sse.uni-hildesheim.de/svn/Indenica/deliverables/wp1/d121-release>
- Unzip the file
- Open the Eclipse Install dialog (Help → Install New Software...)
- Load the IRET update site from the local folder (see Figure A.1)

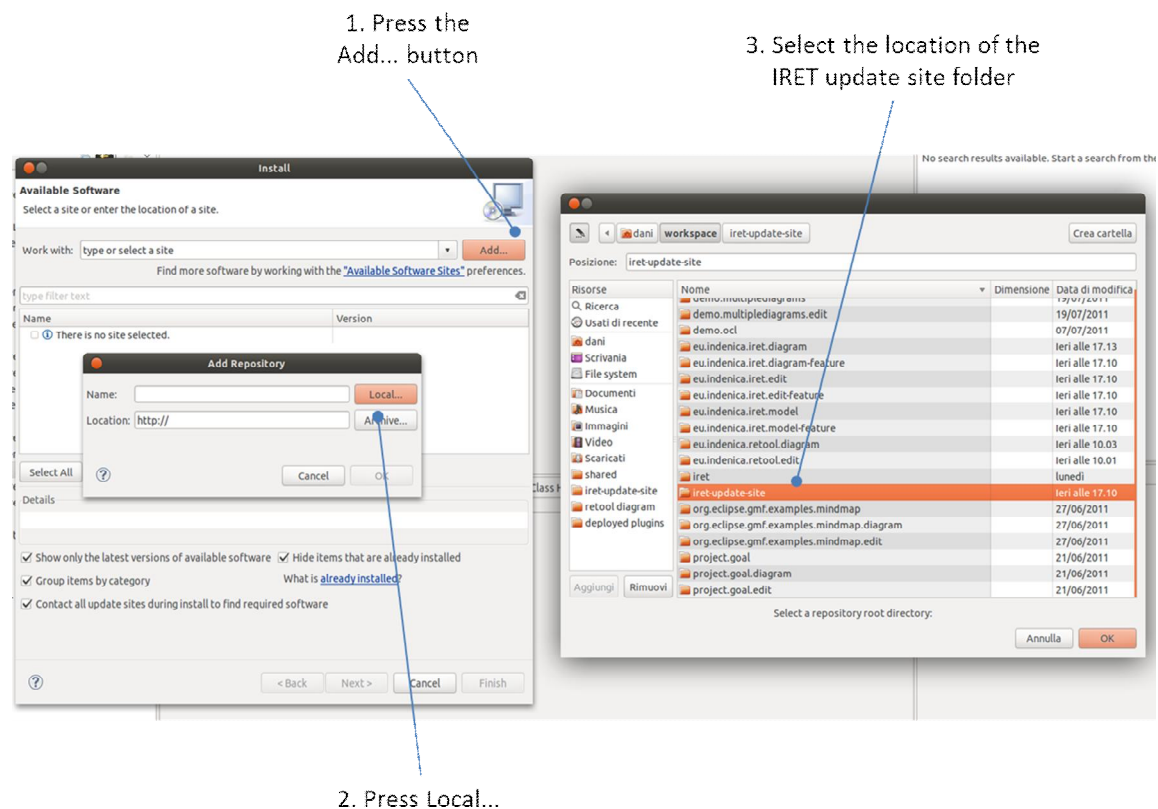


Figure A.1: Load of the IRET update site in Eclipse.

- Check IRENE Toolset to install all the plug-ins composing IRET:
 - o IRET Model
 - o IRET Edit
 - o IRET Diagram
- Press the Next button and follow the wizard procedure: it will install IRET and all the plug-ins it requires to run
- Restart Eclipse